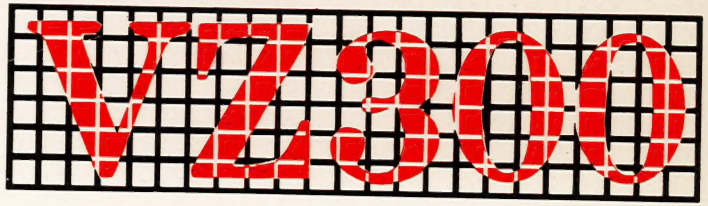
PROGRAMMING

$4.

THE

DICK SMITH ELECTRONICS



Personal Colour
Computer

Tim Hartnell

Programming

the

DICK SMITH ELECTRONICS PTY LTD

VZ300 Personal Colour Computer

Tim Hartnell

Programming the DICK SMITH ELECTRONICS VZ300 Personal Colour Computer

Tim Hartnell

First published in Australia by:
Dick Smith Management Pty. Ltd.,
PO Box 321,
North Ryde, NSW, 2113

First printing 1986
Second printing 1987
Third printing 1988
Dick Smith Catalogue Number: X-7325

The companion volumes in this series are *Programming The VZ300, The Giant Book of Games for the VZ300* and *The Amazing VZ300 Omnibus.*

This book was prepared for Dick Smith Management Pty. Ltd. by Interface Publications (Australia) Pty. Ltd., Chelsea House, 34 Camp Street, Chelsea, Vic., 3196. Any queries regarding the contents of the book should be sent to this address.

Special thanks to the staff of Dick Smith Electronics in Frankston, Victoria, for their assistance in preparing this book.

Artwork by Fin Bristow

National Library of Australia Card No & ISBN 0 949772 37 2

The author, Melbourne-born Tim Hartnell, first met Dick Smith in London in the late seventies while Dick was on a trip to buy the helicopter which he used for his record-breaking world solo flight.

The two kept in touch, and after Tim's return from the UK at the end of 1982, Dick called him one day and asked him to come up to Sydney to see a new product which Dick Smith Electronics had developed. It was the VZ200, the forerunner of the VZ300. Tim was impressed with the machine and its potential, and rang the editor of *Australian Personal Computer* about it. The editor decided the launch of the VZ200 was sufficiently important to change, at the last minute, the proposed cover of the next issue of the magazine, and replace it with a photograph of the VZ200. Tim's review of the computer was the lead story in that issue.

Tim and a couple of authors who had worked with him previously on books published by Tim's company, Interface Publications, then wrote three books on the VZ200, which were distributed exclusively through Dick Smith outlets. The feedback on those books was extremely positive, so it was decided that the VZ300 also deserved dedicated books, such as the one you are reading now.

Tim Hartnell originally left Australia in the middle of 1977 for the UK for what was planned to be a six month's working holiday. He finally stayed for nearly six years, and while in London founded his company, Interface Publications, which now has offices in London, Melbourne and New York. Tim has written fifty or so books (he says he doesn't know the exact number), predominately on computers and related subjects, and they have been translated into eleven languages.

Although he now makes his home in Melbourne, he travels back to the USA and the UK two or three times each year to keep in touch with developments in the computer field. "It gets harder to leave each time," he said recently. "Australia is really the only country I ever want to live in. And with developments like the VZ300, it makes sense professionally, as well as personally."

CONTENTS

Chapter Six — Changing in Mid-Stream

Chapter Seven — Getting into Music

Chapter Eight — A Game and a Test

Chapter Nine — Stringing Along

Chapter Ten — Reading DATA

Chapter Eleven — Getting Listed

Chapter Twelve — Using the Graphics

Chapter Thirteen — Animation

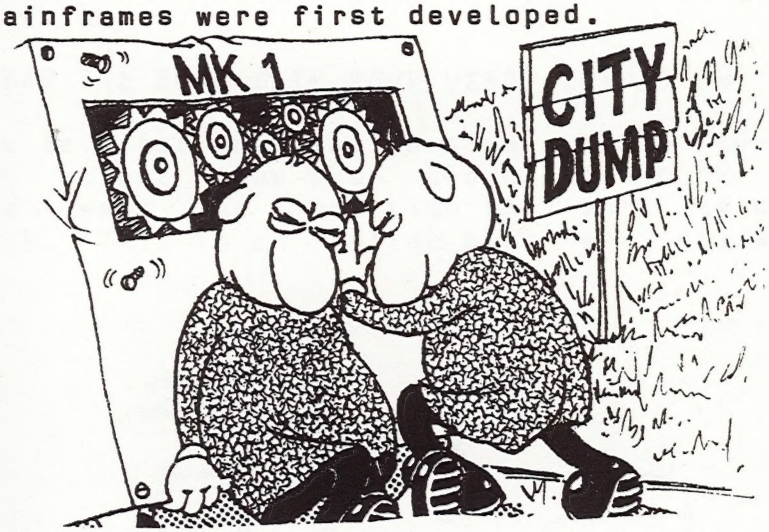
Chapter Fourteen — PEEK and POKE

Chapter Fifteen — Structured Programming

Chapter Sixteen — Cassettes and Disks

Forword

You've just bought a great computer. The VZ300 was a wise buy. It shows just how far computers have come since the old giant mainframes were first developed.



If you've never programmed a computer before this, and you'd like to be able to program the VZ300 in just a few hours, then this is the book for you.

With the help of this book, you'll find out the most important words for programming the VZ300. And, once you've worked through the book, you'll have a library of interesting programs to keep you, and your computer, occupied for weeks to come.

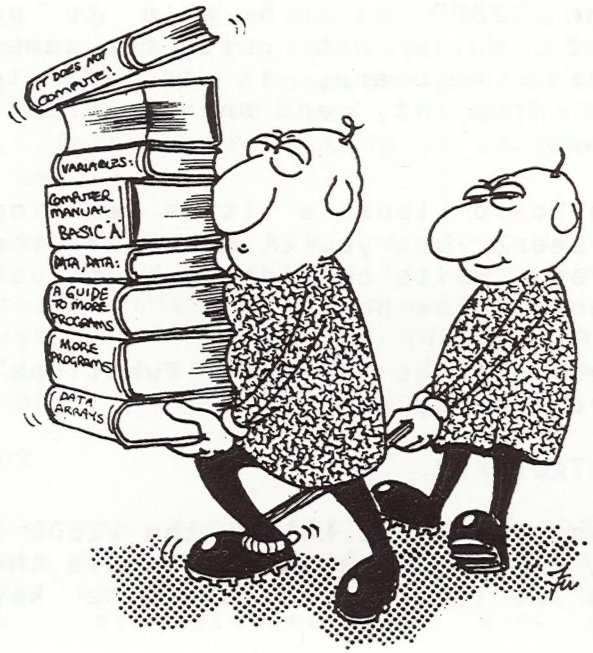
Tim Hartnell,
Melbourne, 1986

Chapter One — Getting Started

Don't worry if the VZ300 is the very first computer you have ever owned. We're going to take things slowly, and in small steps, so you should have no trouble in keeping up with me.

READ THE BOOK WITH YOUR VZ300 TURNED ON

It is vital that you have your VZ300 turned on at all times when reading this book (or at least the first time you work through it). This is an 'action book' and, unlike a



novel, it is not designed simply to be read. This book is like a book on how to drive. You could never learn about changing gears, and how it feels to handle a car when crossing the Harbor Bridge, without actually going out and taking command of a vehicle.

So it is with your VZ and this book. Try out each new command and function when it is explained to you, experiment with the games and other programs, and you'll find you're learning to program without even trying.

THE KEYBOARD

The first thing we have to do in learning to use the VZ300 is to be able to use the keyboard. While not quite the same as a typewriter keyboard, it is not too far removed from it, and should present few problems.

The keyboard looks a little daunting when first seen, but you'll discover that you will feel quite confident about using it after only a few hours.

We'll now look at the major functions of the keyboard step by step.

THE CONTROL KEY

The most important key on the VZ300 is the control key. It is located above the SHIFT key on the left hand side of the keyboard,

and is marked CTRL. This key gives you access to the 'keywords' and 'functions' (which are written on the keys, above and below the main letter on each key). The CTRL key also controls the VZ's editing features.

Holding down the control key, and pressing practically any other key, will give you the keyword written above the main letter written on the key.

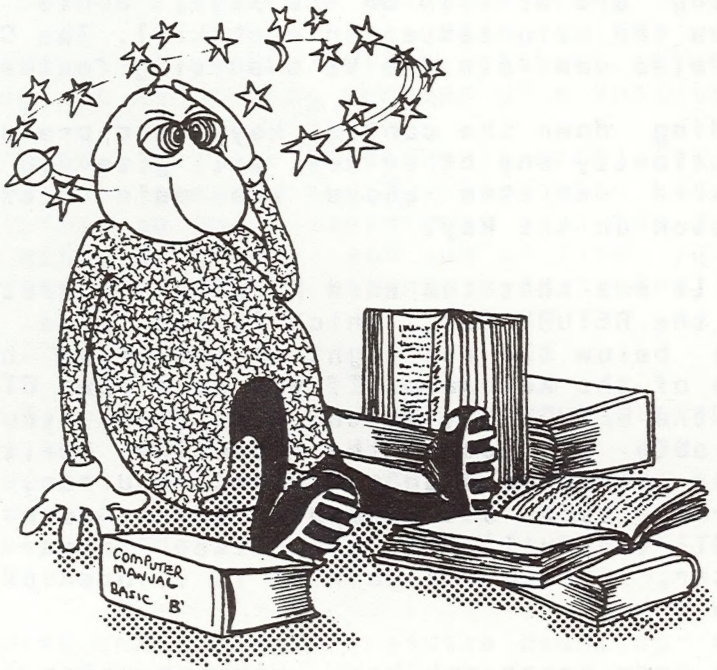
You'll see that the word FUNCTION is written on the RETURN key, which is the wide key just below the red light on the right hand side of the keyboard. If you hold down CTRL, and the RETURN key at the same time, you'll be able to access the functions written below the letters on the keys. You can, of course, type out computer words such as PRINT in full, if you like, letter by letter, instead of getting it in one press from a key.

When you press any key, without using the control key, then you'll get the main character on that key. As well, on most of the keys you'll see a square which is partially, or completely, filled-in. These are graphic characters.

KEYWORDS

Keywords are important parts of programs. They are the sections of the program which actually tell the VZ300 what to do. The keywords are divided into two groups,

commands and functions. We'll look at the commands first.



Many of the commands your VZ300 obeys are like ordinary Aussie words. PRINT, for example, means just what it says, and tells the VZ300 to print something on your TV screen.

Program lines are numbered, as you can see if you look at some of the program listings in this book, and the command GOTO (followed by a number) tells the VZ300 - naturally enough - to GO TO that line number. You'll be pleased to discover that you already know

a large number of words used in the programming language BASIC, which is the language the VZ speaks.

RUBOUT and INSERT

These two keys - located towards the right of the third row of keys (on the 'L' and ';' keys) - are part of the VZ300's editing system. INSERT makes room in a program line for a new character or keyword, and RUBOUT removes a part of a program line which is no longer wanted.

INSERT works by placing a space between any two characters in a line, so that you can later type in new letters in that space. To use it, you just move the cursor (that black rectangle you can see on the screen) over the character which is where you want to place your new material, and press the INSERT key, over and over again, while holding down CTRL key with your left hand, until you have as many spaces as you need.

To use the RUBOUT key, you must move the cursor to the beginning of the character or keyword you want to delete. Hold down CTRL and RUBOUT, and the line will disappear leftward 'under' the cursor.

To get to where you want to in a program listing on the screen, use the 'M', '<', '>' and the space bar, to move the cursor in the direction indicated by the arrows printed on the keys.

SHIFTING

There is, as I'm sure you've noticed, more than one character on most of the keys on your VZ. The symbols in the top right hand corner of each of the keys are punctuation marks, and graphic or arithmetic symbols. You get these characters by holding down the SHIFT key and pressing the relevant key at the same time.

FUNCTIONS

Functions are the second group of keywords (counting the commands, which we've just looked at, as the first group). The functions are, for the most part, written below the main letter on each key. Functions are the 'thinking keywords'. That is, they are often called upon to make decisions while a program is running.

Here's a simple example of a function in use:

```
10 PRINT RND(100);:GOTO 10
```

If you typed this into your VZ, pressed the RETURN key afterwards, then typed in the word RUN (or get the word RUN by holding down the CTRL key and pressing the '6' key), and pressed the RETURN key again, you'd see numbers between 1 and 100 appear all over the screen. Press the BREAK key, in the top right hand corner of the keyboard, while

holding down the CTRL key, to stop the program running.

THE CURSOR

That flashing black rectangle which follows you as you type on the screen is called, as I pointed out a short time ago, the cursor. It has a number of uses.

Because it can move to any part of the screen, it is very useful when you want to edit, or change, part of a program listing. For example, if you wanted to change the 100 in the one-line program given earlier into a 50, you would first move the cursor to the beginning of the number 100. You do this with the versatile CTRL key, using the arrow on the full stop key, until the cursor is at at the beginning of line 10. Then, still keeping CTRL down, you press the comma key until you are over the 1 in 100.

Keep it pressed. If you go past the 1, don't worry. Just use the 'M' key until you are over it again. Still keeping the CTRL key down, press RUBOUT once. The number should now be 00. Type in 5, which will cover the first 0. Then, let go of the CTRL key and press RETURN over and over again, until the cursor is below the program line, and you get the message:

```
?OUT OF DATA ERROR
```

Now type in RUN, and press the RETURN key

again, and numbers between 1 and 50 should appear on the screen.

THE GRAPHICS MODES

We'll only touch on this subject briefly here, and then go into more detail in a later chapter. Your VZ300 can operate in two graphics modes. The first one - MODE 0 - is the standard display mode we've been using to date. The second - MODE 1 - is a graphics mode, in which you can use coloured dots (called 'pixels' in computer-talk) to create pictures.

To see MODE 1 in action, enter and run the next program:

```
10 REM MODE DEMONSTRATION
20 MODE (1)
30 COLOR RND(4),1
40 X=0:Y=0
50 FOR N=1 TO 55
60 SET (X,Y)
70 X=X+1:Y=Y+1
80 NEXT N
90 GOTO 30
```

To get back to MODE 0, hold down CTRL and press the BREAK key.

INVERSE

The INVERSE command, located on the colon (:) key, is used to provide inverse video (that is, light writing on a dark back-

ground) for anything held within quote marks. (Note that your trusty VZ will not accept keywords written in inverse.)

Suppose, for example, you wanted to have a line like the following in a program, and you wanted to have the words within quote marks in inverse letters:

```
10 PRINT "THIS IS THE VZ300"
```

You put your computer into inverse after the first quote mark by holding down the CTRL key and pressing the INVERSE (:) key, then type what you want. Follow the same procedure before you type in the second quote mark to turn the video back to normal.

Keep practicing on the VZ300 until you're used to the keyboard. You'll discover that the more you use your VZ, the more you'll learn about its abilities and capabilities. Don't be afraid to experiment. You can't damage the VZ just by typing in commands.

Well, now that you're familiar with the keyboard, it's time to go to chapter two, and start learning to use the VZ300 itself.

Chapter Two — First Steps in Programming

Time now to take your first steps in programming. You'll be extremely pleased to see how simple and undemanding these first steps will be.

We'll start learning to program the trusty VZ300 by using the most commonly-used command in BASIC, the keyword PRINT.

Type the following into your VZ:

```
PRINT 2 (Note that you can
        either spell PRINT
        in full, or use the
        CTRL key)
```


Until you press the RETURN key, the VZ300 will do nothing. Specifically, at this point, it will ignore the command PRINT 2. Press RETURN now, and you should see the number 2 appear underneath the words PRINT 2. This, pretty obviously, is the way PRINT works. It takes the information which follows the command PRINT, with a few exceptions which we will learn about shortly, and PRINTs this on the screen. This is, after all, exactly what you would expect it to do.

But your VZ is not completely stupid. That is, it can do more than just blindly print what you tell it to do. If the word PRINT is followed by a sum, it will work it out before printing, and given you the answer.

Try it now. Enter the following line, then press RETURN:

```
PRINT  5 + 3 (You get + by
              typing SHIFT ;)
```

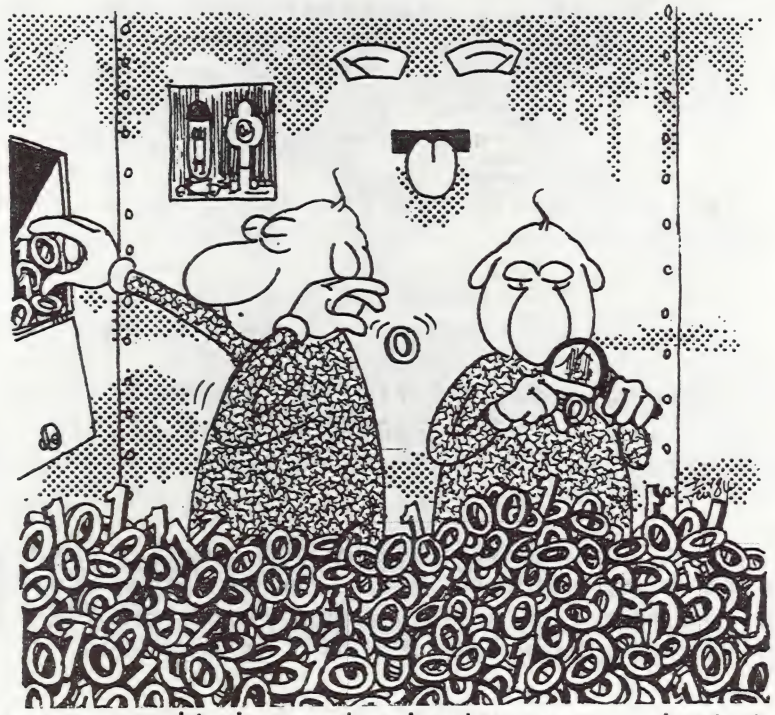
You should see the figure 8 appear. The VZ300 added 5 and 3 together, as instructed by the plus sign, then printed the result on the screen.

It can do subtraction as well (clever inventions, these Dick Smith computers). Type in the following, and press RETURN to see subtraction (and PRINT) at work:

```
PRINT 7 - 2
```

Now the VZ can, of course, carry out a wide range of mathematical tasks, many of them far more sophisticated that simple addition and subtraction. And it is pretty simple, as we've seen to tell the VZ what to do.

It was not so simple in the early days of computers. In those days, the electronic brains did not understand English, or BASIC, and to program them, you had to talk to them in binary (all 0's and 1's). If there was a mistake in a program, a vast number of 1's and 0's had to be checked.



However, it is much simpler now. Just tell the VZ you wish to add, multiply, divide or

subtract, and it will do it for you virtually instantly. There is one tiny hitch. When it comes to multiplication, the computer doesn't use the X symbol you probably used at school. Instead, it uses an asterisk (*) for multiplication, and a slash (/) for divide.

MORE THAN ONE THING AT ONCE

The VZ300 is not limited to a single operation in a PRINT statement. You can combine just about as many as you like. Try the next one, which combines a multiplication and a division. Type it in, then press RETURN to see the VZ work it out:

```
PRINT 5*3/2
```

This seems pretty simple. Just type in the word PRINT, follow it with the information you want the VZ to print, and that's all there is to it.

But it is not quite as simple as that! Try the next one, and see what happens:

```
PRINT TESTING
```

That doesn't look too good when you try it, does it? Instead of the word TESTING, we got a zero. That's because the VZ300 thought we wanted a 'variable', rather than the word TESTING. Now, I'm not going to try and explain the meaning of the word variable at this stage. We'll get to it later. Briefly

though, it means simply that the VZ thought you wanted to print a number which had the name of TESTING.

Silly machine. Computers may be fairly clever, but most of the time they need to be led by the hand, like a very stupid human being (such as myself). Give a computer the right directions, and it will carry it out tirelessly and perfectly, without intervention from the human operator.

But give them incorrect instructions, or - even worse - confuse them, and they give up in despair, or do something quite alien to your intentions.

STRINGS

If you want the computer to print the word TESTING, you have to put quote, or speech, marks around it, like this:

```
PRINT "TESTING"
```

This time when you press the RETURN key, the word TESTING will appear below the line.

Information held in this way between quote marks is called a most peculiar name in computer circles. The jargon for the information enclosed in the quote marks is 'string'. So, in our example above, the word TESTING, when enclosed in quote marks, is a string. (You can, in fact, get away with

just the first pair of quote marks so the line reads PRINT "TESTING but this is not good practice.)

OUR FIRST PROGRAM

Type the following into your VZ300. Notice that each line starts with a number. Type the number into the computer, and follow this with the other material:

```
10 PRINT "JACK AND JILL"
```

Now press RETURN. You have just entered the first line of your first program.

Type in the next line, the one starting with 20, and press RETURN once you have it in place. Do the same with the rest of the lines:

```
20 PRINT "WENT UP THE HILL"
30 PRINT "TO FETCH A PAIL"
40 PRINT "OF WATER"
```

When you RUN this (by typing in RUN, and then pressing the RETURN key), you should see the following:

```
JACK AND JILL
WENT UP THE HILL
TO FETCH A PAIL
OF WATER

READY
```

ADDING NEW LINES

The VZ300, clever beast that it is, allows you to enter your lines in any order you choose. It will then sort them into the correct order for you.

Although our first program, and many of the other ones in this book are numbered in 10's, starting at 10, there is no particular reason why you should follow this convention if you do not want to do so.

However, there is a good reason for leaving 'gaps' in the counting. Although our first program could easily be numbered in 1's, it would leave no room to add later lines, if we decided there was a need to do so.

To see that the VZ does automatically put its lines in order, add the following:

```
25 REM A LINE IN THE MIDDLE
```

Now type in LIST and press RETURN. The word LIST tells the VZ to list out the current program it is holding. When it does so, you'll see line 25 neatly in its numerical place. Now run the program again. You should find that line 25 made no difference at all to it.

MAKING REMARKS

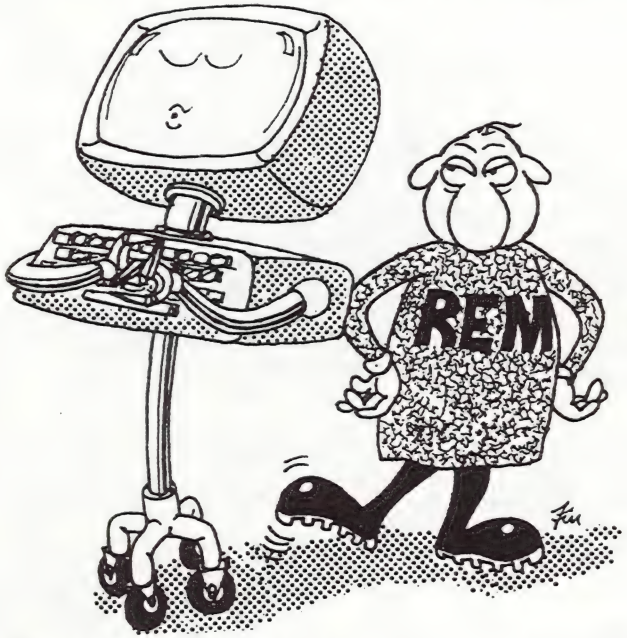
Why did the VZ decide to ignore line 25? The word REM stands for REMARK, and is used within programs when you want to include information for a human being who is looking through the program listing.

You'll find REM statements scattered throughout the programs in this book. In each and every case, the computer ignores them. They are only there for your convenience, for the convenience of the programmer or for anyone else who wants to read through the program listing.

Often you'll use REM statements at the beginning of the program, such as the following:

```
5 REM A JACK AND JILL POEM
```

You may wonder why this should be necessary. After all, it is pretty obvious that the computer is holding a 'Jack and Jill' poem, even without the line 5 REM statement. You are right. In this case, there is little point in adding a title REM statement to the program. But have a look at some of the more complicated programs a little further on in this book. Without REM statements you'd have a pretty difficult time trying to work out what the program was supposed to do.



It is worth getting into good programming habits early in your computer career, so I suggest that, right now, you decide to use REM statements in all your programs (assuming you have memory spare for them).

BACK TO PRINT

Let's return to the subject of the PRINT command. Empty your VZ's memory by typing in the word NEW and press RETURN. Type the following program into the VZ300 and run it:

```
10 PRINT 1,2
20 PRINT 1;2;3
30 PRINT "COMPUTER"
40 PRINT "23+34 =";23+34
50 PRINT 2*3
60 PRINT 3^5
70 PRINT "THE ANSWER IS";23+5-7/6
```

The funny little upside-down 'v' in line 60, by the way, is on the keyboard as a little arrow. You get it from the 'N' key, by holding down SHIFT while you press 'N'.

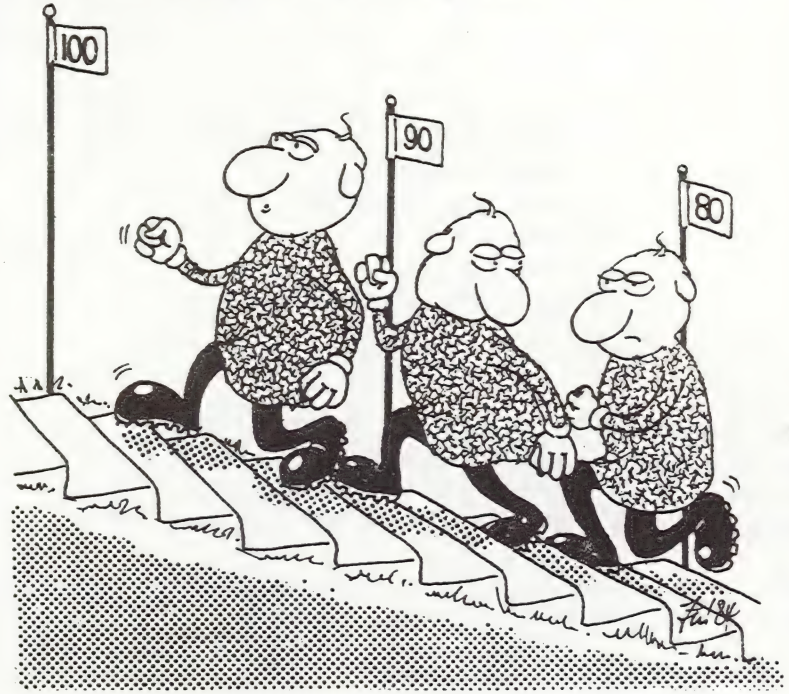
When you run it, you'll see something like this on the screen:

```
 1             2
 1  2  3
COMPUTER
23+34 = 57
 6
 243
THE ANSWER IS 26.8333

READY
```

Firstly, as in the JACK AND JILL program,

the VZ executes a program line by line, starting at the lowest numbered one and proceeding through the line numbers in order, until it runs out of numbers, when it stops. Programs tend to run from the lowest line number up to the highest.



You'll discover that this orderly progression of line numbers does not always apply, as there are ways of making the VZ execute parts of a program out of strict numerical order, but for the time being it is best to assume that the program will be executed in order.

Look first at line 10 of the program. You can see that there is a comma between the 1 and the 2. This has the effect of making the computer print the numbers with a wide space between them. The comma can be used in this way to space numbers out neatly for a table of results or for similar purposes. (Try PRINT 1,,2 and see what effect this has.)

When you use a comma in this way, to divide the things which follow a PRINT statement (but not when the comma is part of a string, that is, is between quote marks), you'll find it divides the screen up into neat little columns. Try PRINT 1,2,3,4,5,6,7,8,9 and see the result of the commas. Then you can try PRINT 1,,,,2,,,,3,,,,4 to make it perfectly clear what is going on.

Line 20 has three numbers (1, 2 and 3) separated not by commas (as in line 10) but by semicolons (;). Instead of separating the output of the numbers as the comma did, you'll see that it causes them to be printed with a single space on each side of them. When printed, numbers are always followed by a space. Positive numbers are also preceded by a space.

Line 30 is a word, and this is a .... If you mentally said 'string' when you came to those dots, then you're learning well. This word is a string, in computer terms, because it is enclosed within quote marks.

Line 40 is rather interesting. For the first

time we have included numbers and a symbol (=) within a string. As you can see, the VZ prints exactly what is within the quote marks, but works out the result of the calculation for the material outside the quotes, giving - in this case - the result of adding 23 to 24.

Keep in mind that the VZ300 considers everything within quote marks as words, even if the material within the quotes is numbers, symbols, or even just spaces, or any combination of them, while it counts everything which is not within quotes in a PRINT statement as a number.

This is why the VZ made a mistake earlier when we told it to print the word TESTING without putting the word in quotes. It looked for a number which was called TESTING, and because it could not find one (as we had not previously told the computer to let TESTING equal some number), it was unable to cooperate.

So line 40 treats the first part, with quote marks, as a string, and the second part of the line, outside the quote marks, as numerical information to process.

In line 50 we see the asterisk (*) used to represent multiplication, and the VZ300 quite reasonably works out what 2 times 3 is and prints the answer of 6. In line 60, we come across a new and strange sign, ^. This

means 'raise to the power', so line 60 says print the result of 3 raised to the fifth power. In ordinary arithmetic, we indicate this by putting the 5 up in the air beside the three. However, it is pretty difficult for a computer to print a number halfway up the mast of another number, so we use the ^ symbol to remind us (by pointing upward) that it really means 'print the second number up in the air'.

The final line of this program combines a string ('THE ANSWER IS') with number information (23+5-7/6). You can see that, as expected, the computer works out the sum before printing the answer, and prints the string exactly as it appears in the original program. Look closely at the end of the string. After the closing quote, there is a semicolon which, as we learned in line 20, joins various elements of a PRINT statement together. This semicolon means that the result of the calculation is printed up next to the end of the string.

This brings us to the end of the second chapter of the book. I'm sure you'll be pleased at how much you've learned so far, and are looking forward to continuing your learning. But, now you've earned a break. So take that break, and then come back to the book to tackle the third chapter.

Chapter Three — Ringing the Changes

It's all very well putting things onto the VZ's screen as we learned to do in the last chapter, but from time to time you'll discover we need to be able to get printed material off the screen during a program, to make way for more PRINT statements. We do this with a command called CLS, for CLear the Screen.

CLEAR THAT SCREEN

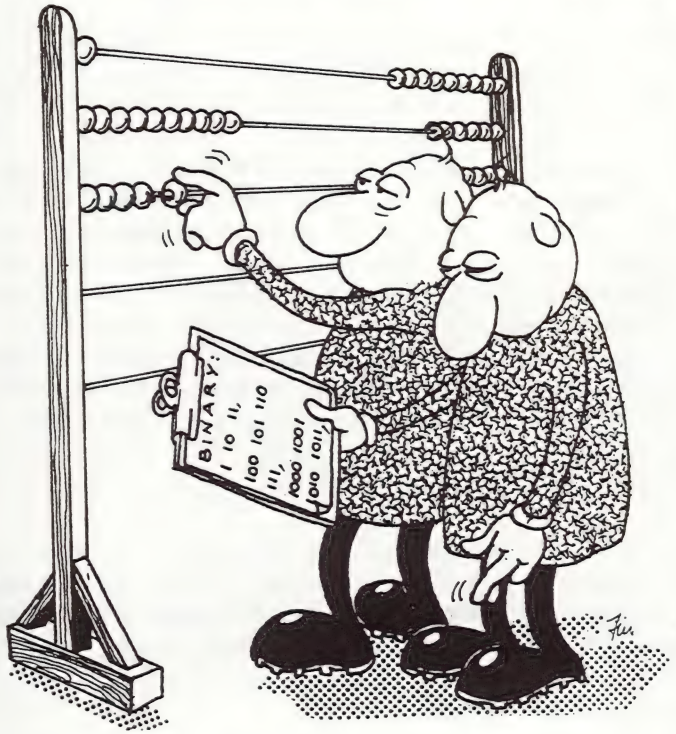
Enter the following program into your VZ300 and run it:

```
10 PRINT "TESTING"
20 INPUT A$
30 CLS
```

When you run the program, you'll see the word 'TESTING' appear under RUN, more or less as you'd expect. However, below it you'll see a question mark. Where did that come from? The question mark is known as an input prompt. An input prompt, which appears in a program which the VZ comes to the word INPUT, means the computer is waiting for you to enter something else into it, or just to press RETURN.

You'll recall those string things we talked about a while ago, and how they were anything which was enclosed within quote

marks. Well, in line 20 above, the computer was waiting for a string input. We know this by the dollar sign which follows the A which follows the word INPUT. If there was no dollar sign, the VZ would want you to enter a number. If there is a dollar sign, it wants a string, or for you just to press the



RETURN key. It doesn't matter which one you enter. The VZ will be happy either way.

When you come to an input prompt (remember, the question mark) and your computer is

expecting a string to be entered, you can either enter word, spaces, graphics characters, numbers, punctuation, or any combination of them. Once you've typed in your words or whatever, you need to press RETURN so that the VZ will know you've finished entering your material.

Anyway, when you respond to the input prompt just by pressing RETURN - rather than enter a word, and then press RETURN - you'll see the screen clears and the word TESTING disappears. Where did it go? We pointed out that the VZ works through a program in line order. Firstly the program printed TESTING on the screen with line 10 and then progressed to line 20, where it waited for an input (or for you to press RETURN). Once you had done this, the VZ moved along to line 30, where it found CLS and obeyed that instruction. The instruction was to clear the screen, so the VZ300 did just that, and the screen cleared.

Run the program a few times, until you've got a pretty good idea of what is happening and you've mentally followed through the sequence of steps the VZ is carrying out.

DOING IT AUTOMATICALLY

Instead of waiting for you to press the RETURN key, you can write a program which clears the screen automatically, as our next tiny program demonstrates. Enter this program into your VZ, type in RUN and then

RETURN, and sit back for the Amazing Flashing Word demonstration. Note that, to make it easy to read what is in a program line, you should put spaces between such words as FOR and TO.

```
10 REM AUTO-CLEAR
20 CLS
30 PRINT @ 234,"AUTOTESTING"
40 FOR D=1 TO 600
50 NEXT D
60 CLS
70 FOR D=1 TO 600
80 NEXT D
90 GOTO 30
```

Run this program, and you'll see the word AUTOTESTING flashing off and on in the middle of the screen.

What is happening here?

We'll look at the program, and go through it line by line. Firstly, as you know, line 10 is a REM statement which labels the program, but which is ignored by the VZ300. Then, line 20 clears the screen to get rid of the program listing.

Line 30 prints the word AUTOTESTING in the middle of the screen. Next, the VZ comes to line 40, where it meets the word FOR. We'll be learning about FOR/NEXT loops (as they are called) in detail in a later chapter,

but all you need to know here is that the computer used FOR/NEXT loops for counting. In this program, lines 40 and 50 (the FOR is in line 40, the NEXT in line 50) to tell the VZ to pause for a while and count from 1 to 600 before moving on. As you can see, it does this counting pretty quickly.

So, it waits for a moment while counting from 1 to 600. Then, it comes to line 60, which is the command CLS, which tells the VZ to clear the screen. The computer then encounters, in lines 70 and 80, another FOR/NEXT loop, so waits a while as it counts from 1 to 600 again.

Continuing on in the program, the VZ comes to line 90 which tells it - as is fairly obvious - to go to line 30, where the whole thing begins again.

SEEING STARS

The following program - STAR COLONY - also shows CLS in action, although it is much more interesting than the AUTOTESTING one.

STAR COLONY is based on the very popular program LIFE, which was first developed by John Conway when he was attending Cambridge University in the UK. The program seeks to simulate the birth, growth and death of a colony of cells. As we are using asterisks to indicate cells, and as these look vaguely like stars, I decided to call this variation of the program STAR COLONY.

Conway drew up the rules under which the stars in the colony evolve. You'll be pleasantly surprised to see just how effective these rules can be when applied, producing unexpectedly delightful designs.

The rules, which are applied by checking the stars surrounding each position on the screen, one by one. As a result of these checks, decisions are made as to whether or not there will be a star in that position the next time the colony is reprinted. If

you imagine that a particular star is sitting on a chess board, somewhere near the centre of the board, you'll see that it has eight squares surrounding it. If, when the contents of the eight surrounding squares are counted, two or three stars are found and there is a star on the square which is surrounded by the other eight, this star will survive until the next generation.

If there are three stars surrounding the square being checked, and the central square is empty, a star will be 'born' in that square in the following generation. If there are four or more surrounding stars, then the central star will die before the next generation is printed.

The rules are applied all over the screen at once, and once the full 'galaxy' has been assessed, the next galaxy generation is printed.

Here is the program listing:

```
20 CLS:PRINT "PLEASE STAND BY"
30 COLOR 2,1
50 DIM A(12,12)
60 DIM B(12,12)
70 G=1
80 FOR X=2 TO 11
90 FOR Y=2 TO 11
110 IF RND(0)>.4 THEN A(X,Y)=1
120 B(X,Y)=A(X,Y)
130 NEXT Y
140 NEXT X:CLS
150 GOSUB 330
```



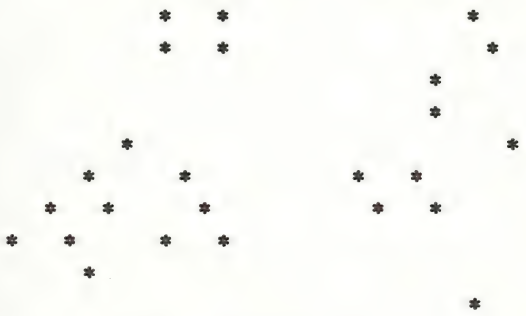
```
160 G=G+1
170 FOR X=2 TO 11
180 FOR Y=2 TO 11
190 C=0
200 IF A(X-1,Y-1)=1 THEN C=C+1
210 IF A(X-1,Y)=1 THEN C=C+1
220 IF A(X-1,Y+1)=1 THEN C=C+1
230 IF A(X,Y-1)=1 THEN C=C+1
240 IF A(X,Y+1)=1 THEN C=C+1
250 IF A(X+1,Y-1)=1 THEN C=C+1
260 IF A(X+1,Y)=1 THEN C=C+1
270 IF A(X+1,Y+1)=1 THEN C=C+1
280 IF A(X,Y)=1 AND C<>3 AND C<>2 THEN B(X,Y)=0
290 IF A(X,Y)=0 AND C=3 THEN B(X,Y)=1
300 NEXT Y
310 NEXT X
320 GOTO 150
330 PRINT @41,"GENERATION"G
340 SOUND RND(25),2
350 PRINT:PRINT "       ";
390 FOR X=2 TO 11
400 FOR Y=2 TO 11
410 A(X,Y)=B(X,Y)
420 IF A(X,Y)=1 THEN PRINT "* "; ELSE PRINT " ";
440 NEXT Y
450 PRINT:PRINT "       ";
460 NEXT X
470 RETURN
```

And here are some 'colonies' produced by it:

GENERATION 1

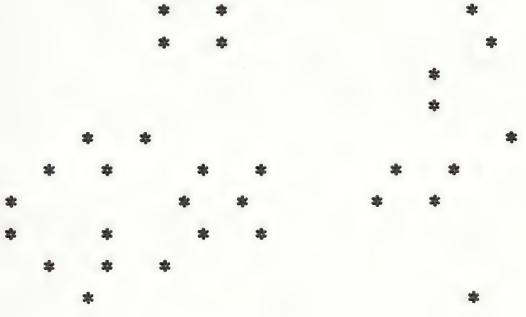
GENERATION 2

GENERATION 12



GENERATION 13

GENERATION 14



CHANGING PROGRAM LINES EASILY

It is relatively easy to change lines within a typed program. All you have to do is use the 'M' and CTRL keys to move the cursor to the required position, then type in the desired changes.

If you have a line 10 which read...

```
10 REM AN EDIT TEST
```

...and you wanted it to read...

```
10 REM AN EXCITING EDIT TEST
```

...all you would need to do would be to move the cursor to the line, then use the INSERT key (the 'L' key, while holding down the CTRL key) to make space for the word EXCITING (nine spaces) before typing it in.

After doing this, you just press the RETURN key again, and type in LIST (or use CTRL 5) and press RETURN again. This time, when the program is listed, you'll see the new version of line 10 is included within the program.

If you only have a single letter wrong within a line, you simply move the cursor to the error, and then type in the correct letter or letters. These will automatically

replace the incorrect material. If you want to wipe out an enter line, just enter the line number, and press RETURN.

Now these instructions may seem a little complex. However, they do not need to be mastered before you can continue your learning.

If you're not sure how a particular line should be edited, and you can't be bothered looking it up here or in your manual, just type in the whole line again. When you press RETURN, the new line will automatically take the place of the old one within the listing.

GETTING THE PROGRAM BACK

If you want to see a complete listing after it has vanished, once a program has been run, all you need to do (as was mentioned briefly before) is type in the word LIST (or use CTRL 5) then press the RETURN key.

There is no reason, when using LIST, why you must list the whole of a program from the beginning. When you have longer programs, you may well want to list only part of them. You do this by use of the hyphen (-), as follows:

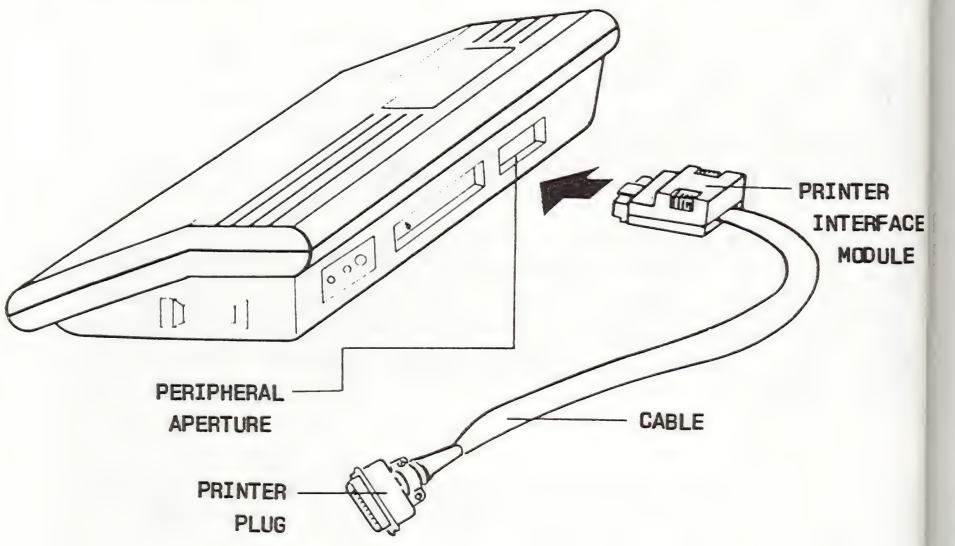
LIST -100	This lists up to, and including, line 100
LIST 50-90	This lists lines 50 to 90

LIST 150- This lists the program from line 150 to the end

LIST 270 This lists just line 270

USING THE PRINTER

You connect your printer to the VZ via the PRINTER INTERFACE MODULE, which you plug in the back of the computer in the PERIPHERAL APERTURE. Make sure you disconnect your VZ from the power before plugging in the interface.



As well, the printer should be disconnected from the power. Once you've removed the cover marked 'PERIPHERAL' from the back, plug the interface module in gently, but firmly. Once it is in place, turn on the VZ and make sure it is working as normal. If it is not, turn the power off, unplug the module, and start again, making sure you are as gentle as possible when sliding it into place.

Once the computer end is OK, plug the other end of the cable into your printer, and turn on the printer.

To test it, just type in the word COPY and press RETURN.

Here are the commands the printer recognizes:

LLIST to list the current program

COPY to print out what is currently on the screen

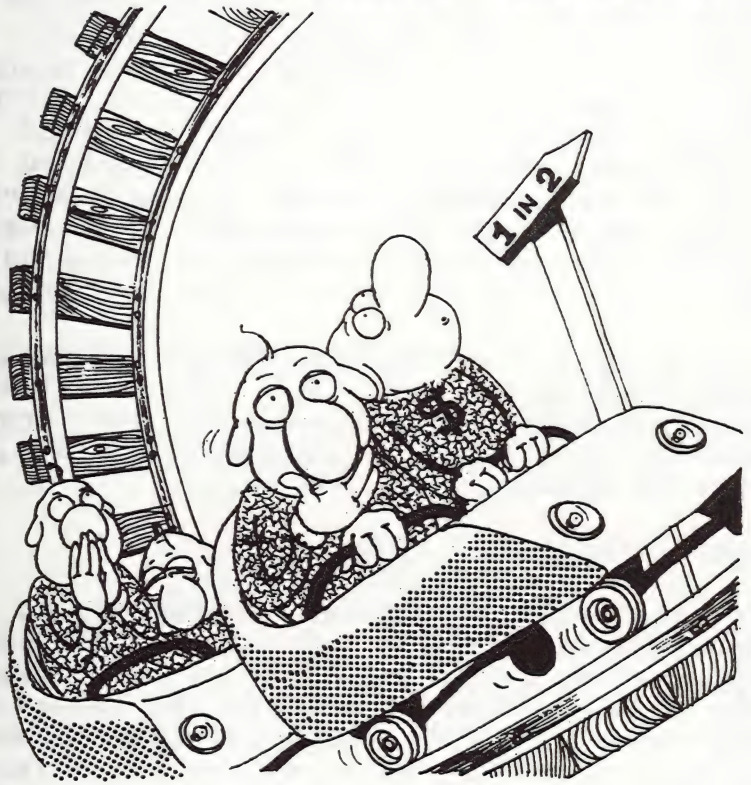
LPRINT used within a program when you want to print something on the printer, rather than on the screen

LLIST is very easy to remember, as it is

very similar to LIST, and has a similar function, except that it lists to the printer rather than listing to the screen. In the same way, LPRINT is easy to remember, as it does more or less what PRINT does, except on paper rather than on the TV. Note that LLIST can be used in the same way as LIST to get just parts of the listing (so LLIST 40 - 70 is a valid command).

Chapter Four — Descent into Chaos

It's time now to start developing some real programs. You'll notice that from this point on in the book there are some lengthy listings. Many of them will contain words from the BASIC programming language which



have not been explained. This is because, as the programs become more complex (and far more satisfying to run) it becomes more and more difficult to keep words - which have

not been explained - out of those programs. However, this is not a major problem, and you'll probably be able to work out what most of them mean, just from seeing them in the context of a program line.

We are working methodically through the commands available on your VZ300, and in due course, all of the important ones will be covered. As we continue up the learning curve, you'll come across words in a program which seem unfamiliar. When you find one like this, just type it in. You'll find that you'll soon start picking up the meaning of words which have not been explained, just by seeing how they are used within the program. So, if you find a new word, don't worry.

The program will work perfectly without you knowing what the word is, and investigating the listing after you've seen the program running is likely to lead you to work out what it means.

RANDOM EVENTS

In the world of nature, as opposed to the manufactured world of man, randomness appears to be at the heart of many events. The number of seagulls over the MCG at any one time, the fact that it rained yesterday in Hobart and may well rain again today, the number of trees growing in the Snowy Mountains, all appear to be somewhat random.

Of course, we can predict with some degree

of certainty whether or not it will rain, but the success of our predictions appears to be somewhat random as well.

When you toss a coin in the air, whether it lands heads or tails depends on chance. The same holds true when you throw a six-sided die down on the table. Whether it lands with the one, the three or the six showing depends on random factors.

Your VZ's ability to generate random number is very useful in order to get the computer to imitate the random events of the real world. The BASIC word RND lies at the heart of using this means of generating random numbers.

GENERATING RANDOM NUMBERS

We'll start by using RND just as it is to create some random numbers. Enter the following program, and run it for a while:

```
10 REM RANDOM DEMO
20 PRINT RND(0);
30 GOTO 20
```

When you do this, you'll see a list of random numbers like these appear on the screen:

```
 .985231   .535247   .0496182   .11153   .514353   .987732   .254916
 .134856   .754681   9.96173E-03   .835601   .31631   .706701   .220855
 .155021   .0714185   .390616   .552147   .173092   .53583   .699251
 .397718   .279562   .398306   .435725   .805982   .295596   .708233
```

As you can see, RND(0) creates numbers randomly between zero and one. It you leave it running, it will go on and on apparently forever, writing up new random numbers on the screen.

Now random numbers between zero and one are

of limited interest if we want to get the numbers to stand for something else. For example, if we could generate 1's and 2's randomly, we could call the 1's heads and the 2's tails, and use the VZ as a kind of 'electronic coin'. If we could get it to produce whole numbers between one and six,

we could use the computer as an imitation six-sided die.

Fortunately, there is a way to do this. Enter the next program and run it:

```
10 REM DICE
20 PRINT RND(6);
30 GOTO 20
```

When you run this program, you'll get a series of numbers, chosen at random between 1 and 6, like these:

```
6 6 6 4 3 4 5 1 6 2 3 3 5 1 6
2 5 6 1 6 5 4 5 4 4 5 5 3 6 2
4 2 3 5 2 4 5 4 6 1 4 3 5 5 1
2 1 3 5 2 1 5 4 4 3 6 5 2 6 6
1 6 2 1 2 2 5 2 1 1 4 5 6 5 2
2 1 4 1 3 1 4 5 5 1 3 6 5 3 6
2 6 1 6 4 5 1 6 1 4 5 2 2 1 6
```

Even though we could create vast series of numbers between 1 and 6 with a program like this, it is not particularly interesting. And, if you ran the program over and over again, you'd find that the sequence of numbers was starting to look very familiar. The random numbers, as you'd discover if you ran the program a number of times, are not really random at all.

This is because the VZ does not really generate random numbers, but only looks as if it is doing so. Inside its electronic

head, your VZ300 holds a long, long list of numbers, which it prints in order when asked for random numbers. The list, fortunately for us, is so long, that it is more or less impossible to see a pattern in it, once it is running.

FAST FOOD CRAZINESS

We'll look now at a program which makes an interesting use of the VZ's ability to generate random numbers. As you can see, it creates a scene where you have turned up at a fast food outlet, unsure of what you are going to order. You decide you'll let your VZ's random number generator pick your food for you.

Just type in this program, and see what you're having for lunch:

```
10 REM FAST FOOD
20 CLS
30 A=RND(4)
40 PRINT "YOU'VE ORDERED ";
50 IF A=1 THEN PRINT "A HAMBURGER"
60 IF A=2 THEN PRINT "A THICKSHAKE"
70 IF A=3 THEN PRINT "CHIPS"
80 IF A=4 THEN PRINT "A HOT DOG"
90 FOR D=1 TO 600:NEXT D
100 PRINT
110 GOTO 30
```

When you run this, you'll get something like the following on your screen:

```
YOU'VE ORDERED A THICKSHAKE

YOU'VE ORDERED A HOT DOG

YOU'VE ORDERED A THICKSHAKE

YOU'VE ORDERED A HAMBURGER

YOU'VE ORDERED CHIPS
```

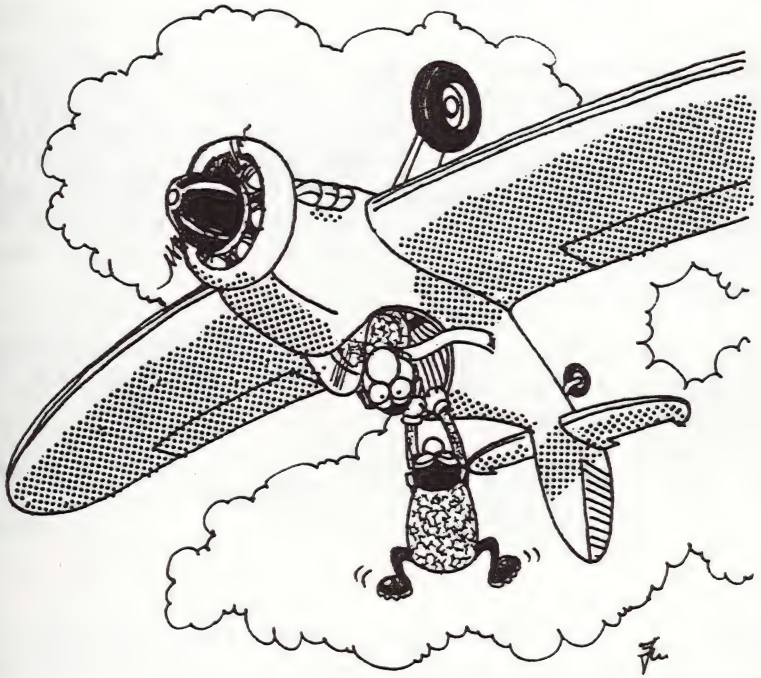
When you look back at the listing, you'll see how the program sets the letter A to the value of the random number in line 30. In this case, the letter A is standing for a

number. It is called a variable, or (because in this case it stands for a number) it is called a numeric variable. In computer jargon, we say that in line 30, the VZ has assigned the value of the random number to the variable A.

And, as you can see in lines 50, 60, 70 and 80, the value assigned to A determines which food order you place.

Chapter Five — Looping the Loop

In this chapter, I'll be introducing a very useful part of your programming vocabulary - FOR/NEXT loops. You'll recall that we



mentioned FOR/NEXT loops when demonstrating the use of CLS to clear the screen. A FOR/NEXT loop was also used in our FAST FOOD program (line 90) to add a delay.

A FOR/NEXT loop is pretty simple. It takes the form of two lines in the program, the

first of which is like this:

```
10 FOR A=1 TO 20
```

With the second line like this:

```
20 NEXT A
```

The control variable, the letter after FOR and after NEXT, must be the same. (You can, in fact, leave the second A out altogether, as the VZ300 will know what you mean. However, leaving the control variable out makes programs harder to read and alter, so this practice is not recommended in your early programming days.)

As a FOR/NEXT loop runs, the computer counts from the first number up to the second, as the next program shows:

```
10 FOR A=1 TO 20
20 PRINT A;
30 NEXT A
```

When you run it, you'll see the numbers 1 to 20 appear on the screen, much as you may have expected.

Now try this version:

```
10 FOR A=765 TO 775
20 PRINT A;
30 NEXT A
```


This is the result of running it:

```
 765  766  767  768  769  770  7
71  772  773  774  775
```

STEPPING OUT

In the two previous examples, the computer has counted up in ones, but there is no reason why it should always count in this way. The word STEP can be used after the FOR part of the first line as follows:

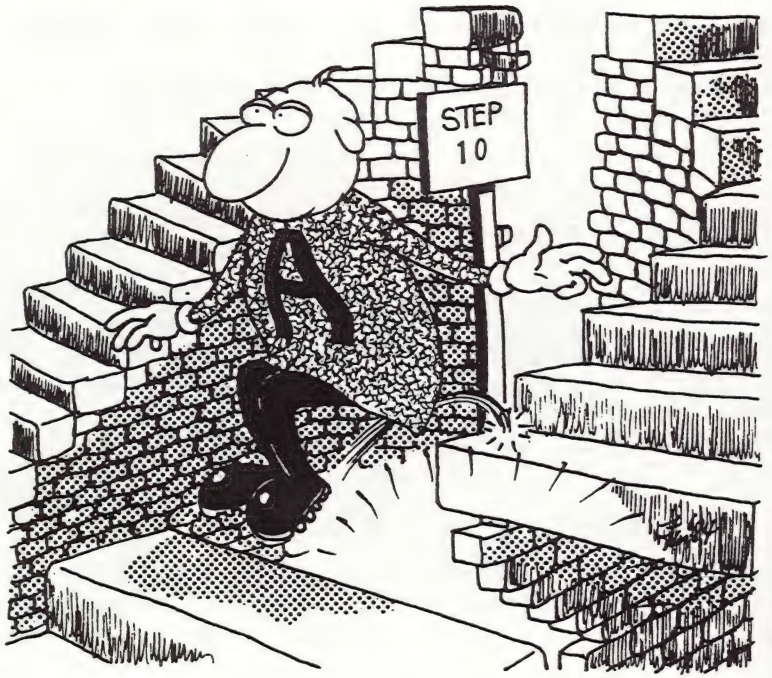
```
10 FOR A=10 TO 100 STEP 10
20 PRINT A;
30 NEXT A
```

When you run this program, you'll discover it counts (probably as you expected) in steps of 10, producing this result:

```
10  20  30  40  50  60  70  80
90  100
```

STEPPING DOWN

The STEP does not have to be positive. Your computer is just as happy counting backwards, using a negative STEP size:




```
10 FOR A=100 TO 10 STEP -10
20 PRINT A;
30 NEXT A
```

This is what the program output looks like:

```
100  90  80  70  60  50  40  30
 20  10
```


MAKING A NEST

It is possible to place one or more FOR/NEXT loops within each other. This is called nesting loops. In the next program, the B loop is nested within the A loop:

```
10 FOR A=1 TO 3
20 FOR B=1 TO 2
30 PRINT A;"TIMES";B;"IS";A*B
40 NEXT B
50 NEXT A
```

The nested program produces this result:

```
1 TIMES 1 IS 1
1 TIMES 2 IS 2
2 TIMES 1 IS 2
2 TIMES 2 IS 4
3 TIMES 1 IS 3
3 TIMES 2 IS 6
```

You need to be very careful to ensure that the first loop which is started is the last loop which is finished. That is, if FOR A... was the first loop you mentioned in the program, the last NEXT must be NEXT A.

Try swapping line 10 with line 20 in the program, and see what happens when you get your FORs and your NEXTs mixed up.

You may recall that I mentioned you do not, in fact, have to mention the control variable with the NEXT if you do not want it. I also said that it was not particularly good programming practice to leave it out as it made programs somewhat difficult to unravel. However, as I imagine you've realised by now, leaving off the control variables at least gets around the problem of wrongly specifying the NEXT in nested loops.

You can replace lines 40 and 50 of the program with either of the following (removing the old line 50 completely):

```
40 NEXT B:NEXT A

40 NEXT:NEXT

40 NEXT B,A
```

MULTIPLICATION TABLES

You can use nested loops to get the VZ to do such fascinating things as printing out the multiplication tables, from one times one right up to twelve times twelve. Wow! Here's the program you need:


```
10 FOR A=1 TO 12
20 FOR B=1 TO 12
30 PRINT A;"TIMES";B;"=";A*B
40 NEXT B:NEXT A
```

Here's part of the output of that program:

```
 7 TIMES 8 = 56
 7 TIMES 9 = 63
 7 TIMES 10 = 70
 7 TIMES 11 = 77
 7 TIMES 12 = 84
 8 TIMES 1 = 8
 8 TIMES 2 = 16
 8 TIMES 3 = 24
 8 TIMES 4 = 32
 8 TIMES 5 = 40
 8 TIMES 6 = 48
 8 TIMES 7 = 56
```

There is no reason why both loops should be travelling in the same direction (that is, why both should be counting upwards) as this variation of the program demonstrates:

```
10 FOR A=1 TO 12
20 FOR B=12 TO 1 STEP -1
30 PRINT A;"TIMES";B;"=";A*B
40 NEXT B:NEXT A
```

Here's part of the output of that program:

```
6 TIMES 7 = 42
6 TIMES 6 = 36
6 TIMES 5 = 30
6 TIMES 4 = 24
6 TIMES 3 = 18
6 TIMES 2 = 12
6 TIMES 1 = 6
7 TIMES 12 = 84
7 TIMES 11 = 77
7 TIMES 10 = 70
7 TIMES 9 = 63
7 TIMES 8 = 56
```

BECOMING A MASTER-MIND

It is time for our next real program. This is a computer version of **Mastermind**, in which your VZ thinks of a four-digit number, such as 5928, and you have eight guesses in which to work out what the number is. You not only have to work out the four numbers the computer has chosen, but also determine the order they are in.

After each guess, the VZ300 will tell you how near you are to the final solution. A 'white' is the right digit in the wrong position, and a 'black' is a correct digit in the right position within the four numbers of the code. As you can see, you are aiming to get four blacks. Digits may occur more than one in the four numbers (so a code of 4426 or even 8888 is possible).

Enter the program into your VZ300 and play a few rounds against the computer. Then, return to the book for a discussion on it, which will highlight the role played by the FOR/NEXT loops.

Here is the listing:

```
10 REM MASTER MIND
20 CLS
30 PRINT @64,"*******************************"
40 PRINT "WHEN YOU ARE TOLD TO DO SO,","ENTER A 4-DIGIT NUMBER"
50 PRINT "AND THEN PRESS <RETURN>."
60 PRINT:PRINT "DIGITS CAN BE REPEATED"
70 PRINT:PRINT "YOU'VE 8 GOES TO CRACK MY CODE"
80 PRINT "*******************************"
90 FOR J=1 TO 1000:NEXT J
100 DIM B(4),D(4)
110 H=0
120 FOR A=1 TO 4
130 B(A)=RND(9)
140 NEXT A
150 FOR R=1 TO 8
160 PRINT:PRINT "THIS IS GUESS NUMBER";R
170 PRINT:INPUT "WHAT IS YOUR GUESS";G
180 IF G<1000 OR G>9999 THEN 170
190 P=INT(G/1000)
200 Q=INT((G-1000*P)/100)
210 T=INT((G-1000*P-100*Q)/10)
220 S=INT(G-1000*P-100*Q-10*T)
230 D(1)=P:D(2)=Q:D(3)=T:D(4)=S
240 FOR E=1 TO 4
250 IF D(E)<>B(E) THEN 300
260 PRINT " BLACK";
270 B(E)=B(E)+10
```

```
280 D(E)=D(E)+20
290 H=H+1
300 NEXT E
310 IF H=4 THEN 510
320 FOR F=1 TO 4
330 D=D(F)
340 FOR X=1 TO 4
350 IF D<>B(X) THEN 390
360 PRINT " WHITE";
370 B(X)=B(X)+10
380 GOTO 400
390 NEXT X
400 NEXT F
410 FOR X=1 TO 4
420 IF B(X)<10 THEN 440
430 B(X)=B(X)-10
440 NEXT X
450 H=0:PRINT
460 FOR D=1 TO 1000:NEXT D
470 PRINT "------------":NEXT R
480 PRINT:PRINT "YOU DIDN'T WORK IT OUT!"
490 PRINT "THE CODE WAS";B(1);B(2);B(3);B(4)
500 GOTO 530
510 PRINT:PRINT "WELL DONE, MASTER MIND!"
520 PRINT "YOU DID IT IN";R;"GOES"
530 PRINT:INPUT "ANOTHER GAME (Y OR N)";A$
540 IF LEFT$(A$,1)="Y" THEN CLS:GOTO 110
550 PRINT "OK BUDDY. THANKS FOR THE GAME"
```


Here's what the screen looks like at the beginning of a run:

```
*******************************
WHEN YOU ARE TOLD TO DO SO,
ENTER A 4-DIGIT NUMBER
AND THEN PRESS <RETURN>.

DIGITS CAN BE REPEATED

YOU'VE 8 GOES TO CRACK MY CODE
*******************************
```

And here's part of one game in progress:

```
 BLACK BLACK
------------

THIS IS GUESS NUMBER 5

WHAT IS YOUR GUESS? 6977
 BLACK BLACK BLACK
------------

THIS IS GUESS NUMBER 6

WHAT IS YOUR GUESS? 6947
 BLACK BLACK BLACK BLACK
WELL DONE, MASTER MIND!
YOU DID IT IN 6 GOES

ANOTHER GAME (Y OR N)? N
OK BUDDY. THANKS FOR THE GAME
```

We'll now go through the program, line by line, a practice we'll be following in several of the programs in this book. If you don't want to read the detailed explanation now (and there may well be parts of it which are a little hard to understand at the moment), by all means skip over the explanation and then come back to it later when you have a bit more background knowledge to work with.

Lines 30 and 80 print a number of asterisks to rule of the title and instructions. Line 90 pauses for a few seconds so you can read this, and then the screen is cleared.

Arrays are dimensioned in line 100. We'll be discussing arrays in a later chapter. For now, all you need to know is that by saying DIM B(4) you tell the computer you want to create a list of objects, with the list called B, in which the first item can be referred to as B(1), the second as B(2), and so on. You do not really need to dimension an array when there are 11 or less items in your list, but it helps to keep your thinking and programs clear if you do always dimension an array before using it in any program.

In Mastermind, arrays are used for storing the number picked by the computer, and for storing the digits which you use each guess when you try to break the code.

H is a numeric variable (we've met them

before) which is set equal to zero in line 110. In line 290, 1 is added to the value of H each time a black is found, so if H ever gets to equal 4, the computer knows all the digits have been guessed, and goes to the routine from line 520 to print up the message about your victory.

The lines from 120 to 140 work out the number which you have to try and guess. Line 110 uses the RND function we've discussed before to get four random numbers in the range 1 to 9, and stores one each in the elements of the B array. Note that the first FOR/NEXT loop of our program appears here. The A in line 120 equals 1 the first time the loop is passed through, 2 the second time and so on, so that the A in 130 changes as well.

Our next FOR/NEXT loop, which uses R, starts in the next line. It counts from 1 to 8, to give you eight guesses. Line 170 accepts your guess, after the previous line has told you which guess it is you are entering.

The numeric variable G is set equal to your guess, and line 90 checks to make sure you have not entered a five-digit number, or one which has less than four digits. If you have done either of these things, the program goes back to line 170 to ask you once again to enter your guess.

The next section of the program, right through to line 450, works out how well

you've done, using a number of FOR/NEXT loops (240 to 300, 320 to 400, 340 to 390 and 410 to 440). Line 470 sends the program back to the line after the FOR R=... to go through the loop again. If the R loop has been run through eight times, then the program does not go back to line 150, but 'falls through' line 470 to 480 to tell you that you have not guessed the number in time, and to tell you what it is. Line 490 prints out the four-digit code.

If you do manage to guess it, so that H equals 4 in line 310, then the program jumps to line 510 to print out the congratulations message.

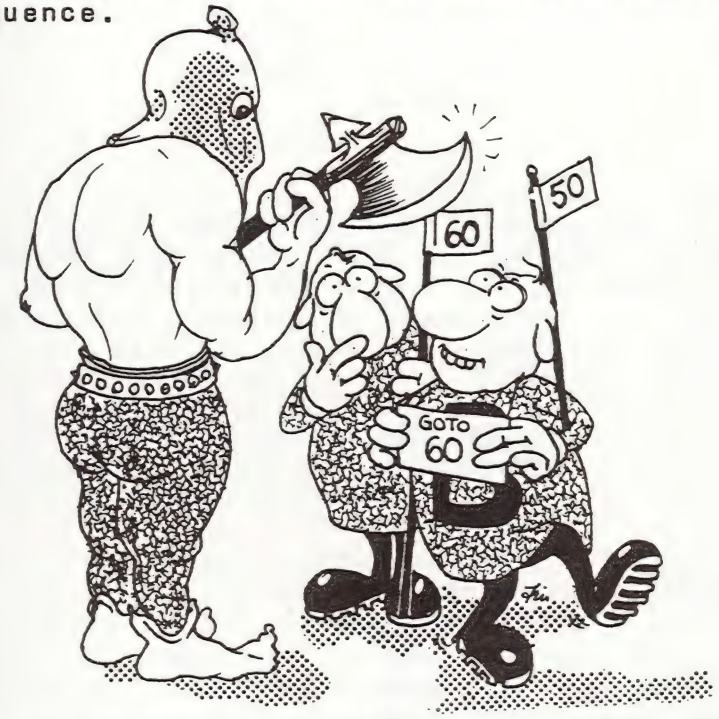
Lines 530 to 560 provide the finishing touch. Line 530 asks you if you would like to play again. If you do, then the computer goes back to 110 for another game.

Even if you don't guess the four numbers in your eight goes, you're offered another go. By returning to line 110, rather than starting the whole program again, you don't have to read through the instructions again before playing the game.

Chapter Six — Changing in Mid-Stream

I pointed out at the beginning of the book that, in most situations, your VZ300 follows through a program in line order, starting at the lowest line number and following through in order until the program reaches the final line.

This is not always true. The command GOTO allows you to break the orderly execution sequence.



As you'll see from our next program, you can use GOTO to send action through a program in

any order you choose. Enter this program, and before you run it, see if you can predict what the result of running it will be:

```
10 REM GO TO
20 GOTO 50
30 PRINT "THIS IS 30"
40 GOTO 70
50 PRINT "THIS IS 50"
60 GOTO 30
70 PRINT "THIS IS 70"
80 FOR Z=1 TO 200:NEXT Z
90 GOTO 50
```

This rather pointless programs sends your poor VZ jumping all over the place, changing its position in the program every time it comes to a GOTO command. Here's what you'll see on your screen when you run it:

```
THIS IS 50
THIS IS 30
THIS IS 70
THIS IS 50
THIS IS 30
THIS IS 70
THIS IS 50
THIS IS 30
THIS IS 70
THIS IS 50
THIS IS 30
THIS IS 70
```


The program ignores the first line, the REM statement, then comes to line 20. Here it finds the command GOTO 50 and when it obeys it, it finds the instruction to print the message "THIS IS 50". It then continues on to line 60, where it is told to GOTO 30. Without question, it zips back to line 30 to print out "THIS IS 30", then goes to line 40 which directs it to GOTO 70. At line 70 it finds the instruction to print out "THIS IS 70" which it obeys. There is a short delay (line 80) then line 90 tells the VZ to GOTO 50 which it does, and the whole thing starts again.

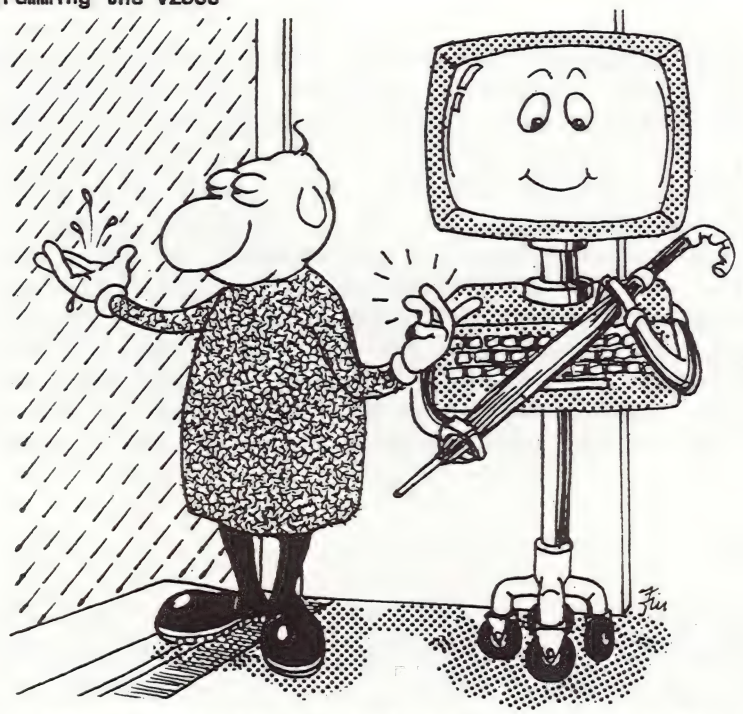
RESTRICTIVE PRACTICES

Making use of GOTO in this way is called **unconditional branching**. The command is not qualified in any way, so the VZ always obeys it.

This brings us to the next computer words we will consider. These are a pair of words, IF and THEN, which are nearly always found together. They impose conditions on branching by GOTO commands.

They are an easy pair of words to understand. IF something is true, THEN do something else. IF you are hungry, THEN order a hamburger. IF it is raining, THEN get an umbrella. IF you want a big car, THEN start saving. IF something, THEN something.

The next program, which 'rolls a die' (using



the random number generator) and then prints up the result of that die roll as a word, uses a number of IF/THEN lines:

```
10 REM DICE ROLL
20 GOTO 140
30 PRINT "ONE"
40 GOTO 140
50 PRINT "TWO"
60 GOTO 140
70 PRINT "THREE"
80 GOTO 140
90 PRINT "FOUR"
100 GOTO 140
110 PRINT "FIVE"
```



```
120 GOTO 140
130 PRINT "SIX"
140 A=RND(6)
150 FOR Z=1 TO 200:NEXT Z
160 IF A=1 THEN 30
170 IF A=2 THEN 50
180 IF A=3 THEN 70
190 IF A=4 THEN 90
200 IF A=5 THEN 110
210 IF A=6 THEN 130
```

This is what you'll see when you run the program:

```
SIX
FOUR
ONE
SIX
FOUR
FIVE
TWO
SIX
TWO
TWO
ONE
FIVE
```

So, we've looked at non-conditional and condition GOTO's to send action all over the place within a program.

SUBROUTINES, ANOTHER WAY TO FLY

There is another way to redirect the computer during the course of a program. This is by the use of subroutines. A subroutine is part of a program which is called twice or more during the course of

running the main part of the program, and is more efficiently kept outside the main program than within it.

The next program should make it clearer. In it, the computer throws a die over and over

again. The first time it is thrown, the computer is throwing it for you. The second time, it throws the die for itself.

After each pair of dice has been thrown, the VZ300 will announce who is the winner. The higher number wins.

The program uses a subroutine to throw the

die, so we do not need two identical 'die-throwing routines' within a single program. Enter and run the program, then return to the book, and I'll explain where the subroutine is within the program, and how it works:

```
10 REM DICE SUBROUTINES
20 GOSUB 210
30 PRINT:PRINT
40 FOR R=1 TO 2
50 GOSUB 150
60 IF R=1 THEN A=D
70 IF R=2 THEN B=D
80 NEXT R
90 PRINT TAB(5);">> ";
100 IF A>B THEN PRINT "VZ 300 WINS"
110 IF A<B THEN PRINT "HUMAN WINS"
120 IF A=B THEN PRINT "IT'S A DRAW!"
130 SOUND 25,1:SOUND5,1:SOUND 25,1
140 RUN
150 REM DICE ROLL SUBROUTINE
160 D=RND(6)
170 PRINT TAB(4);
180 IF R=1 THEN PRINT "  I ROLLED A";D
190 IF R=2 THEN PRINT "YOU ROLLED A";D
200 RETURN
210 REM DELAY SUBROUTINE
220 FOR P=1 TO 400:NEXT P
230 RETURN
```

This is what you'll see when you run it:

```
 I ROLLED A 3
YOU ROLLED A 6
 >> HUMAN WINS


 I ROLLED A 1
YOU ROLLED A 4
 >> HUMAN WINS


 I ROLLED A 5
YOU ROLLED A 5
 >> IT'S A DRAW!
```

The program pauses for a short while on line 220, returns to near the start of the program, prints two blank lines, then enters the R FOR/NEXT loop. When it gets to line 50, which it does once each time through the R loop, the program is sent to the sub-routine starting at line 150.

Here, the 'die is rolled' in line 160, and the numeric variable D is set equal to the result of the roll, using an IF/THEN to determine whether the computer should print out either "I ROLLED A..." or "YOU ROLLED A...". Then the VZ300 comes to the word RETURN in line 200. The word RETURN signals to the computer that it must return to the line after the one which sent it to the subroutine.

In this program, line 50 sent it to the sub-routine, so it returns to line 60.

There, the IF/THENs in lines 60 and 70 determine whether the value of the roll (variable D has been set equal to this value) should be assigned to variable A or to variable B.

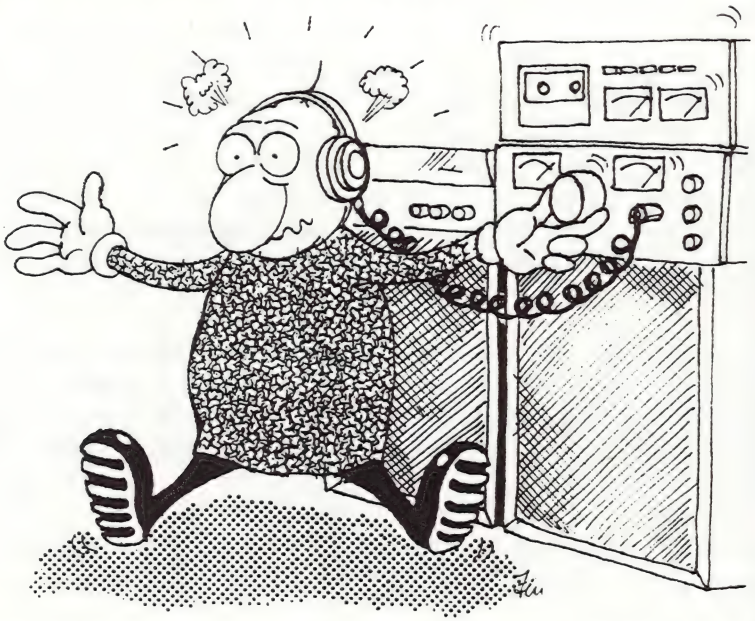
Line 80 ends the FOR/NEXT loop, and then lines 100 through to 120 determine whether the VZ has won (which it will have done if A is greater than B, a condition which is tested using the > sign in line 100), or whether the human has won (which happens if A is less than B, a condition tested in line 110 with the < symbol).

From here, the program RUNs again from the beginning. Remember, to stop the VZ300 running an 'endless' program of this type, you hold down the CTRL key, and press BREAK until the program halts.

Study this last program, until you're pretty sure you know how subroutines work.

Chapter Seven — Getting into Music

The SOUND command is a great way to add life to your programs. It is amazing, as you'll soon discover, what a small amount of sound can do to enhance a program.



SOUND is always followed by two numbers (or by variables representing numbers). The first number is the pitch, or frequency, of the note to be played, and the second determines for how long the note will sound. The pitch is a number between 0 and 31, and the duration is a number between 1 and 9. The pitch value of 0 produces no sound. It can be used as a 'rest', in music terms.

Here's a simple program which puts the SOUND statement through a few of its paces:

```
10 REM SOUND DEMO
20 FOR S=0 TO 31 STEP 2
30 SOUND S,1
40 NEXT S
```

And you can combine more than one SOUND statement at a time within a loop for an even more effective result. All you need to do is add a single line - 35 - to the program above for an incredible change in the effect it produces:

```
35 SOUND 31-S,1
```

THE PHANTOM COMPOSER

The VZ300 can even write its own music - after a fashion. Enter and run the following program and you'll hear the magic for yourself:

```
5 REM PHANTOM COMPOSER
10 S=RND(31)
20 D=RND(2)
25 FOR J=1 TO RND(3)
27 NEXT J
30 FOR J=1 TO RND(3)
40 SOUND S,D
```

```
50 IF RND(0)>.7 THEN SOUND 31-S,D
60 NEXT J
70 GOTO 10
```

The Phantom Composer, as it pretty obvious from the program, choses notes and durations at random, and plays them. From time to time, it will repeat a phrase two or three times (using the loop from 30 through to 60) and sometimes add in an extra note during the repeat (line 50).

To become your own Phantom Composer, and write music on the VZ300, you need to know the exact musical equivalents of the note and duration numbers.

FREQUENCY

Here are the frequencies for the VZ300. The first number is the number you use (as the first one after the SOUND command), and the second is its actual musical value:

0 - REST	1 - A2
2 - A#2	3 - B2
4 - C3	5 - C#3
6 - D3	7 - D#3
8 - E3	9 - F3
10 - F#3	11 - G3
12 - G#3	13 - A3
14 - A#3	15 - B3
16 - C4	17 - C#4
18 - D4	19 - D#4
20 - E4	21 - F4

22 - F#4	23 - G4
24 - G#4	25 - A4
26 - A#4	27 - B4
28 - C5	29 - C#5
30 - D5	31 - D#5

DURATION

Now, here are the relevant note lengths. The first number is the one you have after the comma in the SOUND command, and the second number is its value, taking a quaver to have a value of 1:

Number	*Duration*
1	1/8
2	1/4
3	3/8
4	1/2
5	3/4
6	1
7	1 1/4
8	2
9	3

SOUND ADVICE

The control numbers for the SOUND command, as listed above, can be the result of calculations, instead of being numbers or assigned variables. In the next program, SOUND ADVICE, you have to guess the number between 1 and 50 which the VZ has thought of. The feedback on each guess - which will help you home in on the correct number in

the shortest number of guesses - is in the form of output produced by SOUND.

The lower the note, the closer you are to the correct answer. Once you've played the game a few times, you'll be pleased to see how skilled you become at interpreting the output.

Here's what you see on the screen when playing the game:

```
THIS IS GUESS 5
WHAT NUMBER AM I THINKING OF? 10

NO, 10 IS WRONG

THIS IS GUESS 6
WHAT NUMBER AM I THINKING OF? 4
NO, 4 IS WRONG

THIS IS GUESS 7
WHAT NUMBER AM I THINKING OF? 7

YES! I WAS THINKING OF 7
YOU GOT IT IN 7 TRIES
```

And here's the listing. Note that ABS in line 100 stands for 'absolute' and gives the result of the calculation, stripped of its sign. That means, if the result of a calculation is a negative number, say -7, the ABS of that number is 7, so ABS(-7)=7. The ABS of a positive number is just the number, unchanged.


```
10 REM SOUND ADVICE
20 CLS
30 N=RND(50)
40 R=1
50 PRINT:PRINT "THIS IS GUESS";R
60 INPUT "WHAT NUMBER AM I THINKING OF";GUESS
70 IF GUESS<1 OR GUESS>50 THEN 60
80 IF GUESS=N THEN 120
90 PRINT "NO,";GUESS;"IS WRONG"
100 S=ABS(N-GUESS):SOUND S,2
110 R=R+1:FOR D=1 TO 400:NEXT D:GOTO 50
120 PRINT:PRINT "YES! I WAS THINKING OF";N
130 SOUND 31,1:SOUND 1,2:SOUND 31,1
140 PRINT "YOU GOT IT IN";R;"TRIES"
```

The first line of the program is, of course, just a REM statement to tell you the name of the program. The next line clears the screen, and line 30 sets the variable N equal to a number chosen at random between 1 and 50. This is the number which you have to try and guess.

The variable R is set equal to 1 in line 40. As you've probably realised, this is used to count the number of guesses you have made.

Line 50 prints up the number of the current guess and line 60 asks you to enter a number. The following line checks the size of your guess, rejecting it if it is above 50 or below 1.

Note that line 70 ends with THEN 60, rather than THEN GOTO 60, as you may have expected.

You are allowed to leave out the word GOTO after THEN, as the VZ will understand what you mean. If, however, you feel that the program will be easier to understand with the word GOTO in place, by all means replace it, and do the same in the following line.

Line 80 checks your answer, and if it finds that your guess is the same as the number the computer has thought of, it sends action to line 120 where the congratulations message is printed.

BUT WHAT IF YOU'RE WRONG?

If you're not right, the program goes on to line 90 where the message "NO, x IS WRONG" is displayed. Now we come to the interesting bit. The variable S is set equal to the absolute difference between the computer's number and your guess. The sound is produced at the end of line 100, 1 is added to the value of the variable R in the next line, and then after a pause the program returns to 50 for your next guess.

Loch MacDicksmith

This beautiful lake in northern Scotland is unknown to most tourists. Instead of a monster, Loch MacDicksmith has its very own piper, who plays on that traditional Scottish instrument, the VZ-bagpipes. If you want to hear the wonderful sounds the piper creates, run the next program:


```
10 REM VZ BAGPIPES
20 DIM N(10)
30 FOR S=1 TO 10
40 READ N(S)
50 NEXT S
60 M=RND(10):D=RND(5)
70 SOUND N(M),D
80 FOR J=1 TO RND(2):NEXT J
90 GOTO 60
100 DATA 4,6,8,9,11,13,15,16,16,4
```

MAKING YOUR OWN MUSIC

If the 'auto-bagpipes' are too much for you, try the next program, which allows you to use the bottom row of keys as a kind of organ. It is not too musical, but you should have some fun with it:

```
10 REM VZ 300 ORGAN
20 CLS
30 C$=INKEY$
40 IF C$="" THEN 30
50 IF C$="Z" THEN SOUND 4,2
60 IF C$="X" THEN SOUND 6,2
70 IF C$="C" THEN SOUND 8,2
80 IF C$="V" THEN SOUND 9,2
90 IF C$="B" THEN SOUND 11,2
110 IF C$="N" THEN SOUND 13,2
120 IF C$="M" THEN SOUND 15,2
130 IF C$="," THEN SOUND 16,2
140 IF C$="." THEN SOUND 18,1
150 IF C$=" " THEN END:REM SPACE
160 IF INKEY$<>"" THEN 160
170 GOTO 30
```

To play the organ, just touch the keys on the bottom row of the keyboard, as follows:

Press this key:	*To get this note:*
Z	C
X	D
C	E
V	F
B	G
N	A
M	B
,	C'
.	D'

Press the space bar to end the music.

Chapter Eight — A Game and a Test

It's time now to take a break from the serious business of learning to program. As you can see in this chapter, we have two major programs which use many commands which have not yet been explained. I suggest you enter the programs just as they are and play them for your own enjoyment, and forget about learning for a while.

I don't think it's fair to keep you waiting for major programs until you've covered everything on the VZ300. Also, entering short demonstration programs can get pretty boring if you're longing to see your computer really in action. Therefore, I hope you'll enter the programs 'on trust', returning to this chapter for the explanations when you're ready. Of course, you don't have to enter the programs right now. If you'd prefer to continue with the learning, then just go on to chapter nine.

OUT ON THE FAIRWAY

In the first game, you and your versatile VZ have to tackle the Dicksmith Golfcourse.

You have nine holes to negotiate, and as you'll see when you play the game, the VZ obligingly keeps the score card for you. After each hole, it will tell you how well you're doing to date, and then will work out the average score per hole. All you have to

do is hit the ball. If you overshoot, the VZ will automatically ensure that the next shot is back towards the hole.

Here's what the program looks like in action:

```
SCORE UP TO THIS HOLE IS 6
<< HOLE NUMBER 2 >>
DIFFICULTY FACTOR IS THREE

              O
####################\ /######
#################### #######
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

ENTER STROKE STRENGTH? 12

SCORE UP TO THIS HOLE IS 14
<< HOLE NUMBER 3 >>
DIFFICULTY FACTOR IS FIVE


####################\ /######
#####################O#######
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

ENTER STROKE STRENGTH? 2
YOU DID IT!

THE GAME SO FAR:
HOLE 1 TOOK 6 STROKES
HOLE 2 TOOK 8 STROKES
HOLE 3 TOOK 9 STROKES
THE AVERAGE SO FAR IS 7
SCORE FOR 3 HOLES IS 23
```


You'll probably find it pretty tricky going, especially on holes with a high difficulty factor.

Here's the listing, golf pro:

```
10 REM GOLF
20 DIM X(9):C=0:H$=CHR$(48)
30 U=224
35 L$="                                ":REM 32 SPACES
40 FOR Z=1 TO 9
45 CLS
50 SC=0
60 J=RND(12)
70 Q=RND(3)+2
80 IF Q=5 THEN Q$="FIVE"
90 IF Q=4 THEN Q$="FOUR"
100 IF Q=3 THEN Q$="THREE"
110 PRINT @448,L$:PRINT @64,
115 IF Z=1 THEN PRINT
120 IF Z>1 THEN PRINT "SCORE UP TO THIS HOLE IS";
130 IF Z=2 THEN PRINT X(1)
135 IF Z>2 THEN PRINT K
140 PRINT "<< HOLE NUMBER";Z;">>"
150 PRINT "DIFFICULTY FACTOR IS ";Q$
160 GOSUB 430
170 PRINT
175 INPUT "ENTER STROKE STRENGTH";A
177 SOUND 31,1
180 PRINT @U,L$
185 IF J>24 THEN A=-A
200 J=J+INT(A/RND(Q))
205 IF J=24 THEN GOSUB 490
206 IF J>30 THEN J=30
207 IF J<1 THEN J=1
210 IF J<>24 THEN PRINT @U+J-1,H$
```

```
215 IF J<>24 THEN PRINT@352,L$:PRINT L$
220 SC=SC+1
230 PRINT @448,"YOUR SCORE IS NOW";SC
240 FOR P=1 TO 700:NEXT P
250 IF J<>24 THEN 110
260 C=C+SC
270 X(Z)=SC
280 IF Z=1 THEN 390
290 K=0
300 PRINT "THE GAME SO FAR:"
310 FOR J=1 TO Z
320 K=K+X(J)
330 PRINT "HOLE"J"TOOK"X(J)"STROKES"
340 FOR M=1 TO 300:NEXT M
350 NEXT J
360 IF Z=9 THEN 370
365 PRINT "THE AVERAGE SO FAR IS"INT((K+.5)/Z)
370 FOR P=1 TO 1000:NEXT P
380 IF Z>1 THEN 390
385 PRINT "FIRST HOLE SCORE:"C
386 GOTO 400
390 PRINT "SCORE FOR"Z"HOLES IS"C
400 FOR M=1 TO 2500:NEXT M
410 NEXT Z
420 GOTO 560
430 IF J>30 THEN J=30
435 PRINT @196,""
440 PRINT TAB(J-1);H$
450 PRINT "######################\ /######"
460 PRINT "###################### ######"
470 PRINT "^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^"
480 RETURN
490 PRINT@416,"YOU DID IT!"
500 PRINT@311,H$
510 FOR P=1 TO 300:NEXT P
```



```
520 SOUND 21,4:SOUND 16,2
525 SOUND 16,1:SOUND 18,4
527 SOUND 16,4:SOUND 0,1
530 SOUND 20,4:SOUND 21,4
540 FOR P=1 TO 2000:NEXT P
550 RETURN
560 PRINT "END OF THAT ROUND!"
570 PRINT
575 PRINT "YOU SCORED"C
580 PRINT "& YOUR AVERAGE WAS"INT((C+.5)/9)
590 PRINT:PRINT
600 PRINT "ENTER 'Y' FOR A NEW ROUND,"
605 PRINT "   OR 'N' TO QUIT"
610 A$=INKEY$
620 IF A$<>"Y" AND A$<>"N" THEN 610
630 IF A$="Y" THEN RUN
640 PRINT
650 PRINT "OK, SEE YOU AGAIN SOON"
```

TESTING YOUR SPEED

The second program in this chapter, REACTION TESTER, is great fun to play. You enter the program, RUN it, and the message "STAND BY" appears. After an agonizing wait, the words will vanish, to be replaced by the words "OK, HIT THE 'Z' KEY".

As fast as you can, you have to leap for the 'Z' and press it, knowing that the VZ300 is keeping count of your actions all the time.

The computer will then tell you how quickly you reacted, and then compare this with your previous best time. "THE BEST SO FAR IS..."

appears on the screen, and the VZ then waits for you to take your hands off the keyboard

to prevent cheating (as if you'd do such a thing!) before the whole process starts again.

```
YOUR SCORE IS 22.0224

OK, HIT THE 'Z' KEY!

THE BEST SO FAR IS 13.9986

STAND BY
```


The game continues until you manage to get your reaction time down below 4, which is not at all easy to do.

Here's the listing of REACTION TEST:

```
5 REM REACTION TESTER
7 CLS
10 HS=1000
15 FOR W=1 TO 3000
16 COLOR 3,1
17 IF HS<4 THEN 95
19 CLS
20 PRINT @236,"STAND BY"
25 GOSUB 105
27 COLOR 2,0
30 GOSUB 100
35 IF A$<>"" THEN 25
40 C=0
42 COLOR 1,1
45 PRINT @134,"OK, HIT THE 'Z' KEY!"
50 C=C+1.07+RND(0)
52 PRINT @1,C;"    "
55 GOSUB 100:IF C>=200 THEN 90
60 IF A$<>"Z" THEN 50
65 PRINT
67 PRINT "YOUR SCORE IS";C
70 IF C<HS THEN HS=C:SOUND 30,2
75 PRINT:PRINT:PRINT
77 PRINT "THE BEST SO FAR IS";HS
78 GOSUB 105
80 GOSUB 100
85 IF A$<>"" THEN 80
90 NEXT W
95 PRINT
97 PRINT "YOU'RE THE CHAMP!"
```

```
98 SOUND 31,5:SOUND 28,5:SOUND 1,2:SOUND 31,5
99 SOUND 7,7:SOUND 3,1:END
100 A$=INKEY$:RETURN
105 FOR P=1 TO 500+RND(999)
110 NEXT P
115 CLS:RETURN
```

OK, that's enough fun. Time to get back to the hard slog in chapter nine.

Chapter Nine — Stringing Along

You'll recall that several times in this book so far we've referred to numeric variables (which can be letters such as A or B, words like CN or GS, or combinations of letters and numbers such as R2 or D2). We've also talked about string variables (a letter followed by a dollar sign, such as A$ or G$ is a string variable). In this chapter, we'll be looking at strings, and at things you can do with them.

THE CHARACTER SET

Every letter, number or symbol the computer prints has a code number. Telling the computer to print the character of that code produces the character. The codes are known a ASCII (pronounced 'ass-key') codes, which is an almost universal (in the world of computers) code for letters, numbers and symbols, in which a number between 0 and 255 is assigned to each of them. The number 65, for example, always stands for 'A'.

Now, the ASCII code of 'A' is 65. You know this, because I just told you. If I hadn't told you, how would you have found out?

The computer word to find out a code is ASC. Try it now. Enter the following into your VZ300, and see what you get:

```
PRINT ASC("A")
```

Note that the letter for which you want the code must be within brackets, and quote marks as above. When you enter that line on

your VZ, and press RETURN, you should get an answer of 65. (The ASC value of the letter "A" has nothing to do with the value assigned to A when it is a numeric variable. If it is in quote marks, it is a letter, the letter "A". If it is not in quote marks, it is a numeric variable, A.)

From the short demonstration line above we confirmed that 65 was indeed the ASC of "A". We can turn a 65 into the letter "A" by

asking the computer to print the character which corresponds to ASC code 65. We do this with the word CHR$, as follows:

```
PRINT CHR$(65)
```

When you enter this into the old VZ, and press RETURN, the letter "A" will appear on the screen.

You can print out every ASC code and its character with the next short program. Enter it, and watch closely when you run it:

```
10 REM CHARACTERS
20 FOR J=32 TO 255
30 PRINT J;CHR$(J);"   ";
40 FOR D=1 TO 100:NEXT D
50 NEXT J
```

This is the start of what you'll see printed out on the screen:

```
32      33 !    34 "    35 #
36 $    37 %    38 &    39 '
40 (    41 )    42 *    43 +
44 ,    45 -    46 .    47 /
48 0    49 1    50 2    51 3
52 4    53 5    54 6    55 7
56 8    57 9    58 :    59 ;
60 <    61 =    62 >    63 ?
64 @    65 A    66 B    67 C
68 D    69 E    70 F    71 G
72 H    73 I    74 J    75 K
```

```
76 L    77 M    78 N    79 O
80 P    81 Q    82 R    83 S
84 T    85 U    86 V    87 W
88 X    89 Y    90 Z    91 [
92 \    93 ]    94 ^    95 _
96      97 !    98 "    99 #
```

TESTING YOUR CHARACTER

Our next program is a reaction tester, a little like the one we played with in the preceding chapter. However, in this one you're not just being tested on speed. In this program you have to try and find the right key as quickly as possible.

A letter will appear on the screen. As quickly as you can, find that letter on the keyboard and press it. You'll be told how long it took you, and this time will be compared to your best time.

Notice how the letter which is printed on the screen uses CHR$ in line 25, where it prints the character of a number chosen at random by line 15 and assigned there to variable A. A$ is set equal to INKEY$ (which reads the current key you're touching, without you having to press the RETURN key) in line 35 and compared with the letter your VZ has chosen, in line 40.

```
5 REM CHARACTER TEST
6 BEST=200
7 CLS
8 GOSUB 95
```



```
15 A=65+RND(26)
16 IF A=B THEN 15
17 B=A
20 S=0
25 PRINT @423,CHR$(A)
35 A$=INKEY$
40 IF A$=CHR$(A) THEN 70
45 S=S+1
50 PRINT @293,S
60 IF S<200 THEN 35
65 PRINT "SORRY - TIME IS UP"
67 GOTO 80
70 SOUND 30,3
72 PRINT "WELL DONE. YOU SCORED"
75 PRINT "  ";S"ON THAT ONE!"
80 IF S<BEST THEN BEST=S
85 PRINT:PRINT
90 PRINT "BEST SCORE SO FAR";
92 PRINT " IS";BEST
93 IF BEST=3 THEN END
95 FOR P=1 TO 19*S:NEXT P
100 CLS:SOUND 4,4:SOUND 20,3:SOUND 4,2
105 GOTO 15
```

CUTTING THEM UP

One of the very useful aspects of the BASIC used on the VZ300 is the way in which it can be used to manipulate strings.

The words used to handle strings are:

LEFT$
MID$
RIGHT$

These are usually spoken aloud as 'left-string', 'mid-string' and 'right-string'.

Our next program shows them in action. Enter it and run it on the VZ, then return to the book for a brief discussion to see what can be learned from it.

```
5 REM STRINGS
7 CLS
10 A$="THE*ELECTRIC*DICK"
15 PRINT "LEFT$(A$,3)=";LEFT$(A$,3):GOSUB 55
20 PRINT "LEFT$(A$,5)=";LEFT$(A$,5):GOSUB 55
25 PRINT "RIGHT$(A$,3)=";RIGHT$(A$,3):GOSUB 55
30 PRINT "RIGHT$(A$,5)=";RIGHT$(A$,5):GOSUB 55
35 PRINT "MID$(A$,3)=";MID$(A$,3):GOSUB 55
40 PRINT "MID$(A$,5)=";MID$(A$,5):GOSUB 55
45 PRINT "MID$(A$,5,4)=";MID$(A$,5,4):GOSUB 55
50 PRINT "MID$(A$,2,7)=";MID$(A$,2,7):GOSUB 55
52 END
55 FOR J=1 TO 300:NEXT J:RETURN
```

Look at the output.

```
LEFT$(A$,3)=THE
LEFT$(A$,5)=THE*E
RIGHT$(A$,3)=ICK
RIGHT$(A$,5)=*DICK
MID$(A$,3)=E*ELECTRIC*DICK
MID$(A$,5)=ELECTRIC*DICK
MID$(A$,5,4)=ELEC
MID$(A$,2,7)=HE*ELEC
```


As you can see, the program first, in line 10, sets the string variable A$ equal to the string "THE*ELECTRIC*DICK". Then it uses LEFT$, RIGHT$ and MID$ to extract portions of that string.

Look at the first line of the output. LEFT$(A$,3)=THE. LEFT$ takes the leftmost portion of the string, as far as the number which follows the string. That is, when we have LEFT$(A$,3) it takes the three leftmost characters of the string. The next printout, LEFT$(A$,5) takes the five leftmost characters of the string, producing in this case THE*E (because they are five leftmost characters of the overall string).

It can be used slightly differently. If we said...

```
PRINT LEFT$("THE*ELECTRIC*DICK",3)
```

...the VZ would print out THE just the same as it did in the program. The string, then, can either be a string variable (A$) or the string in full ("THE*ELECTRIC*DICK").

As you've probably worked out by now, RIGHT$ does the same thing as LEFT$, except it starts at the righthand end of the string. Therefore, RIGHT$(A$,3) selects the three rightmost characters of the string, in this case, ICK. Again, as above, this is the same as saying:

```
PRINT RIGHT$("THE*ELECTRIC*DICK",4)
```

MID$ is a little more flexible. It selects a portion from the middle of the string, starting from the character number which follows the string. Therefore, MID$(A$,4) prints all the string starting with the fourth character.

If there is only one number, such as the 4 above, then MID$ selects all of the string to the end of it. However, if there is another number, this second number dictates the length of the string which will be extracted.

You can see in the last two printouts from

the program that MID$(A$,5,4) prints the extract of the string four characters long, starting from character five. MID$(A$,2,7) produces a string seven characters long, starting from the second character.

Rerun the program now, putting your name in place of THE*ELECTRIC*DICK in line 10.

PUTTING THEM BACK TOGETHER

Strings can be added together on the VZ. The process of adding strings is called the frightening-looking word **concatenation**. You can concatenate two or more complete strings together, or just add bits of them, as our next program shows:

```
5 REM JOINING STRINGS
7 CLS
10 A$="DICK*SMITH"
15 B$="*AUSTRALIA"
20 C$=A$+B$
25 PRINT "A$=";A$
30 PRINT "B$=";B$
35 PRINT "C$=";C$
40 D=RND(8):E=RND(12)+D
45 PRINT "MID$(C$"D","E")=";MID$(C$,D,E)
50 D$=MID$(C$,D,E)
60 FOR J=1 TO 50:NEXT J
65 GOTO 40
```

When you run this program, which creates C$ in line 35 by concatenating A$ and B$,

you'll see results like these:

```
A$=DICK*SMITH
B$=*AUSTRALIA
C$=DICK*SMITH*AUSTRALIA
MID$(C$ 3 , 6 )=CK*SMI
MID$(C$ 6 , 10 )=SMITH*AUST
MID$(C$ 8 , 10 )=ITH*AUSTRA
MID$(C$ 2 , 5 )=ICK*S
MID$(C$ 1 , 4 )=DICK
MID$(C$ 8 , 14 )=ITH*AUSTRALIA
MID$(C$ 4 , 12 )=K*SMITH*AUST
```

PLAYING AROUND

You can do a number of things with string manipulation, as our next program demonstrates. NAME PYRAMID allows you to enter your name to produce a somewhat interesting display. Once you've seen the program running, you'll understand why it has been given its name.

Here's the listing:

```
10 REM NAME PYRAMID
20 CLS
30 INPUT "WHAT IS YOUR NAME";N$
40 CLS
50 IF LEN(N$)>15 THEN N$=LEFT$(N$,15)
60 FOR L=1 TO LEN(N$)
70 PRINT TAB(16-L);
80 FOR H=1 TO 2*L
90 PRINT MID$(N$,L,1);
100 NEXT H:PRINT
110 NEXT L
```

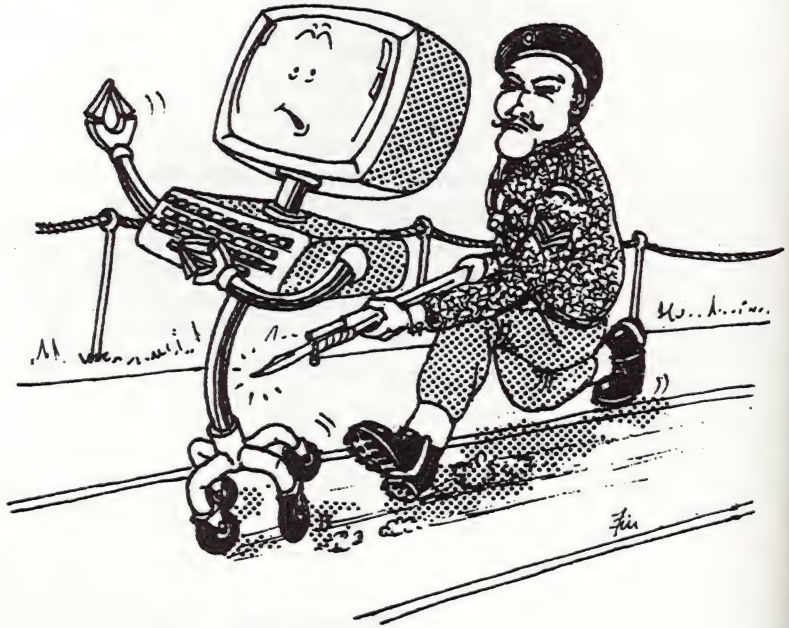

And here are two runs of the program:

```
                DDDD
               IIIIII
              CCCCCCCC
             KKKKKKKKKK
            **********
           **********
          SSSSSSSSSSSSSSS
         MMMMMMMMMMMMMMMMM
        IIIIIIIIIIIIIIIIIII
       TTTTTTTTTTTTTTTTTTTTT
      HHHHHHHHHHHHHHHHHHHHHHH
     ***************************
    *****************************
   *******************************
```

```
              **
             TTTT
            IIIIII
           MMMMMMMM
          **********
         HHHHHHHHHHHH
        AAAAAAAAAAAAAA
       RRRRRRRRRRRRRRRR
      TTTTTTTTTTTTTTTTTT
     NNNNNNNNNNNNNNNNNNN
    EEEEEEEEEEEEEEEEEEEEE
   LLLLLLLLLLLLLLLLLLLLLLL
  LLLLLLLLLLLLLLLLLLLLLLLLL
 ****************************
```

Chapter Ten — Reading DATA

In this chapter, we'll be looking at three very important additions to your programming vocabulary: READ, DATA and RESTORE.



They are used to get information which is stored in one part of a program to another part where it can be used. Enter and run this program to see them in action:

```
10 REM READ/DATA
20 CLS
30 DIM A(5)
40 FOR B=1 TO 5
```



```
50 READ A(B)
60 PRINT A(B)
70 NEXT B
80 DATA 3676,321,-473,4499,33
```

Using line 50, the program READs through the DATA statement in line 80 in order, printing up each item with line 60.

RESTORE moves the computer back to the first item of DATA in a program, as you'll discover if you modify the above program, by adding line 45 to it, so it reads as follows:

```
10 REM READ/DATA
20 CLS
30 DIM A(5)
40 FOR B=1 TO 5
45 IF B=3 THEN RESTORE
50 READ A(B)
60 PRINT A(B)
70 NEXT B
80 DATA 3676,321,-473,4499,33
```

It does not matter where in the program the DATA is stored. The VZ will seek it out, in order from the first item of DATA in the program to the last, as our next program (which scatters the DATA about in an alarming way) convincingly demonstrates:

```
1 DATA 3676
10 REM READ/DATA
20 CLS
```

```
30 DIM A[5]
40 FOR B=1 TO 5
45 DATA 321,-473
50 READ A[B]
60 PRINT A[B]
65 DATA 4499
70 NEXT B
80 DATA 33
```

READ and DATA work just as well with string information:

```
10 REM READ/DATA
20 CLS
30 DIM A$[5]
40 FOR B=1 TO 5
50 READ A$[B]
60 PRINT A$[B]
70 NEXT B
80 DATA VZ300,DICK,SMITH,ELECTRONICS,KANGAROO
```

Note that string DATA does not have to be enclosed within quote marks, unless preceding, or trailing spaces, and/or punctuation and symbols are significant, and must be considered part of the DATA.

You can mix numeric and string DATA within the same program, so long as you take care to ensure that when the program wants a numeric item, a number comes next in the DATA line, and when it wants a string item, it can find what it needs:


```
10 REM READ/DATA MIXED TYPE
20 CLS
30 FOR B=1 TO 5
40 READ A$,A
50 PRINT A$,A
60 NEXT B
70 DATA DICK,23,SMITH,99
80 DATA ELECTRONICS,1231
90 DATA VZ300,4,COMPUTER,111
```

Chapter Eleven — Getting Listed

An array is used when you want to create a list of items, and refer to the item by just mentioning the position within the list which the item occupies. You set up an array by using the command DIM (for dimension).

If you type in DIM A(20), the VZ will set up a list in its memory called A, and save space for twenty-one items: A(0), A(1),

A(2)...and so on...up to A(20). Each of these items - the A(7) and the rest - are called elements of the array.

When you dimension, or set up, an array, the VZ300 creates the list in its memory, and then fills every item in that list with a zero. So if you told your computer to PRINT A(3), it would print a 0.

You fill the elements of an array with values with a statement like A(3)=1000, or by using READ and DATA as we saw in the previous chapter. And once you've given an item a value, you can get the VZ to tell you what value the element has by typing in PRINT A(n). You can also manipulate the element as though it was a number. That is, A(4)*6 is valid, as if 45-A(6) and so on.

The VZ300 will let you use an array of up to eleven elements - that is, A(0) through to A(10) - without having to use a DIM statement first. The moment it comes across the reference to an array, where the subscript (the number which follows the letter name of the array, in brackets, is called the subscript) is between 0 and 10, it automatically creates an array. However, it is good practice to always dimension arrays, even if you're only using subscripts from 0 through to 10.

THE FORGOTTEN ELEMENT

You may like to 'forget' about the element which has the subscript of zero, and pretend that the array starts at one. Many times you'll find it easier, as I always do, to assume that a statement like DIM A(80) gives you array of 80 elements (rather than 81, as is the case), and that the first element is A(1) rather than A(0).

The first program in this chapter dimensions, or sets up, an array called A, with room for 15 elements, although we will ignore the element with the subscript 0. The B loop, from lines 40 through to 60, fills the array with digits between 1 and 9, randomly chosen, and then prints them back for you with the loop from 70 through to 100 (with a slight pause being created by line 90).

Here is the listing:

```
10 REM ARRAYS
20 CLS
30 DIM A(14)
40 FOR B=1 TO 14
50 A(B)=RND(9)
60 NEXT B
70 FOR Z=1 TO 14
75 PRINT TAB(4+(Z>9));
80 PRINT "A(";Z;")  IS ";A(Z)
90 FOR DELAY=1 TO 100:NEXT
100 NEXT Z
```


And here is one example of it in use:

```
A( 1 )  IS  5
A( 2 )  IS  7
A( 3 )  IS  4
A( 4 )  IS  4
A( 5 )  IS  8
A( 6 )  IS  1
A( 7 )  IS  1
A( 8 )  IS  1
A( 9 )  IS  8
A( 10 )  IS  4
A( 11 )  IS  5
A( 12 )  IS  7
A( 13 )  IS  6
A( 14 )  IS  8
```

This is called a one-dimensional array, because a single digit follows the letter of the name which labels the array.

You can also have multi-dimensional arrays, in which more than one number follows the array label after the DIM. In our next program, for example, the VZ sets up a two-dimensional array called A, consisting of five elements by five elements. It is dimensioned by the line DIM A(4,4) as you can see in line 30:

```
10 REM MULTI-DIM ARRAYS
20 CLS
30 DIM A(4,4)
40 FOR B=1 TO 4:FOR C=1 TO 4
```

```
50 A(B,C)=RND(9)
60 NEXT C,B
70 COLOR 8:PRINT "    1  2  3  4"
80 PRINT TAB(3);"##############"
90 FOR B=1 TO 4:PRINT B;"#";
100 FOR C=1 TO 4:PRINT A(B,C);
110 NEXT C:PRINT "#":NEXT B
120 PRINT TAB(3);"##############"
```

When you run it, you'll see something like this:

```
   1  2  3  4
  ##############
1 # 3  9  9  1 #
2 # 7  6  7  3 #
3 # 9  3  6  7 #
4 # 6  5  6  9 #
  ##############
```

You specify the element of a two-dimensional array by referring to both its numbers, so the element 1,1 of this array (the element in the top left hand corner of the printout above) is 3 and is referred to as A(1,1). The 1 in the bottom row is A(4,3) and the 5 next to it is A(4,4).

STRING ARRAYS

Your VZ300 also supports string arrays. Enter and run the following short program to

see string arrays in operation:

```
10 REM STRING ARRAYS
20 CLS
30 DIM A$(5)
40 FOR B=1 TO 5
50 A$(B)=CHR$(RND(25)+65)
60 A$(B)=A$(B)+CHR$(RND(25)+65)
70 NEXT B
80 FOR B=1 TO 5
90 PRINT TAB(4);"A$(";B;") IS ";A$(B)
100 NEXT B
```

Here's one printout of the program:

```
A$( 1 ) IS DZ
A$( 2 ) IS ZM
A$( 3 ) IS YT
A$( 4 ) IS XF
A$( 5 ) IS SY
```

You can, of course, fill the elements of an array, string or numeric, via DATA or INPUT statements. Here is a string array which is filled using DATA:

```
10 REM STRING ARRAYS
20 CLS
30 DIM A$(5)
40 FOR B=1 TO 5
50 READ A$(B)
```

```
70 NEXT B
80 FOR B=5 TO 1 STEP -1
90 PRINT TAB(4);"A$(";B;") IS ";A$(B)
100 NEXT B
110 DATA ELECTRONICS,SMITH,DICK,FROM,VZ300
```

This is the result of running it:

```
A$( 5 ) IS VZ300
A$( 4 ) IS FROM
A$( 3 ) IS DICK
A$( 2 ) IS SMITH
A$( 1 ) IS ELECTRONICS
```

ESCAPE FROM MURKY MARSH

The final program in this chapter demonstrates the use of a two-dimensional array for 'holding' an object, and for moving it around within the array. The shape is trapped in a murky marsh, and by moving totally at random, it hopes one day to be able to escape from the marsh. The shape is free if it manages to stumble onto the outer rows.

The shape in this program, by the way, demonstrates 'Brownian Motion', the random movement shown by such things as tiny particles in a drop of water when viewed under a microscope, or of a single atom of

gas in a closed container. Brownian motion explains why a drop of ink gradually mixes into the water into which it has been placed.

Here is the program listing, showing Brownian Motion in action in the murky marsh:

```
10 REM MURKY MARSH
20 CLS
30 DIM A(10,10)
40 M=0
50 GOSUB 240
60 IF Q>9 OR P>9 THEN 320
70 P=P-1:IF RND(0)>.35 THEN P=P+2
80 Q=Q-1:IF RND(0)>.35 THEN Q=Q+2
90 IF Q<1 THEN Q=1
100 IF Q>10 THEN Q=10
110 IF P<1 THEN P=1
120 IF P>10 THEN P=10
130 M=M+1
140 PRINT @40,"ATTEMPT NO."M"  "
150 A(P,Q)=94
160 PRINT @65,
170 FOR X=1 TO 10
175 PRINT TAB(10);
180 FOR Y=1 TO 10
190 PRINT CHR$(A(X,Y));
200 NEXT Y
210 PRINT
230 NEXT X:SOUND RND(31),1
235 A(P,Q)=254:GOTO 60
240 Q=RND(3)+4
250 P=RND(3)+4
```

```
260 FOR X=1 TO 10
270 FOR Y=1 TO 10
280 A(X,Y)=254
290 NEXT Y,X
300 RETURN
310 REM — THE END —
320 PRINT
330 PRINT TAB(5);"WHEW! FREE AT LAST!"
```


Chapter Twelve — Using the Graphics

Your VZ300 can produce some very effective displays, using the different modes and colours.

THE MODES

As you know, the VZ can operate with two different kinds of display, called modes. The first mode, and the one which is most

frequently used, is mode 0, which is predominately a text mode. The VZ is in this mode when you first turn it on.

You can use numbers, letters and the onboard

'chunky' graphics in mode 0. This mode has a resolution of 31 by 16. That is, the screen can fit 32 characters across and 16 down, as if it was divided into 512 little rectangles, each of which holds one character. Using the chunky graphic blocks - the little squares with bits missing from them on the keys - can give you the effect of producing pictures on a slighly higher resolution, 64 by 32.

The second mode, mode 1, is the high resolution graphics mode. Relatively find individual points can be printed on the screen to create clear pictures. You cannot, however, print text in this mode.

The mode 1 resolution is 128 by 64, with 128 points across and 64 down. The co-ordinates of these points, which are called pixels in VZ-speak, begin in the top left hand corner of the screen, at the point described as the location (0,0).

USING COLOUR

You can use colour in both modes (although the command uses the American spelling of 'color'). The background screen colour can be green or orange. As you know, the screen is green when the computer is first turned on, but it can be changed to orange very simply, just by typing in the following, then pressing RETURN:

```
COLOR,1
```


Note the comma before the number. To turn the screen green again, without turning the computer off, enter this:

```
COLOR,0
```

There are eight colours which can be used in both modes. In mode 0, these colours can only be used on the onboard graphics. High resolution colour is slightly different from that used in mode 0. You have to assign a colour to the plotted points, or you won't be able to see them.

The high resolution background colours are green and grey. You need to use two numbers, as well as the two co-ordinates, to assign a colour to a pixel. Try this:

```
10 MODE(1)
20 COLOR RND(4),1
30 X=RND(120)
40 Y=RND(60)
50 SET(X,Y)
60 GOTO 20
```

The SET function assigns the co-ordinates for a point. The word SET is on the 'H' key. Two numbers, or variables, allow SET to plot a point at the co-ordinates given. RESET clears these points from the screen.

THE TWO MODES

Before we go on to use SET to produce some very interesting displays, you might like to

try the following two programs which make it extremely clear how the two modes differ on the VZ300.

As you can see from the listings, the first one, for mode 0, uses PRINT@ to position the coloured blobs, while the second program uses SET.

```
10 REM TEXT RESOLUTION
20 COLOR,1:CLS:PRINT:PRINT
30 INPUT "WHAT'S YOUR NAME";N$
40 CLS
50 COLOR RND(8)
60 PRINT @(RND(512)),CHR$(141);N$;CHR$(142)
70 SOUND RND(31),1
80 GOTO 50
```

```
10 REM HIGH RESOLUTION
20 CLS
30 INPUT "WHICH BACKGROUND (0 OR 1)";B
40 IF B<0 OR B>1 THEN 30
50 MODE(1)
60 COLOR RND(8),B
70 SET (RND(127),RND(63))
80 GOTO 60
```

SPIROGRAPH PATTERNS

Spirograph patterns are formed by both the interior and exterior epicycloid curves (no, I don't know what that means, either, but it

says so in my encyclopedia). One of the formulae for such a curve is given in line 100 of the following program, in which A is the radius of a large circle, B is the radius of a smaller circle, and H is a point on the circumference of the smaller circle.

The program choses A, B and C at random, then traces out the shape on the screen. The finished pattern is held for a moment, then the process begins again. This program shows clearly how effectively the SET command can be used and produces pictures which are worthy of close examination.

If you don't like a particular pattern which is forming, or you can't see one at all, which will happen rarely with some colour combinations, just press any key and a new design will begin. Some of the results of this program are very, very attractive indeed. You'll find the patterns look best if viewed from a slight distance.

Here's the SPIROGRAPH program listing, so you can get those epicycloid curves to do their thing:

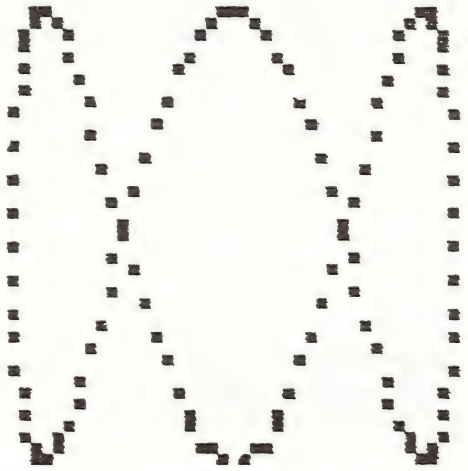
```
10 REM SPIROGRAPH
15 PI=3.141592
20 MODE(1)
25 COLOR INT(RND(0)*3)+2
30 IF INKEY$<>"" THEN 30
40 A=RND(15)
50 B=RND(15)
60 H=RND(15)
70 FOR J=0 TO 3*PI/1.7
80 FOR Q=0 TO 2*PI STEP PI/60
90 X=(A-B)*COS(Q)+H*COS((A-B)*Q/B+J)
95 IF INKEY$<>"" THEN RUN
100 Y=(A-B)*SIN(Q)+H*SIN((A-B)*Q/B+J)
110 SET((60+X),(32+Y))
120 NEXT Q
130 NEXT J
140 FOR T=1 TO 1000:NEXT T
150 RUN
```

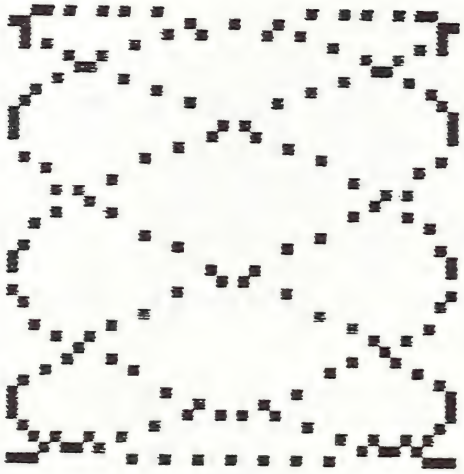

LISSAJOUS FIGURES

Jules Antoin Lissajou, a French physicist who lived from 1822 to 1880, made a study of the movement of particles under the action of periodic motions acting at right angles to each other (at least that sounds a little simpler than those epicycloid curves). Jules discovered that bodies moving in this way trace intricate patterns as they dance around each other.

As in the SPIROGRAPH program we've just looked at, SET is used to plot the points on the screen, tracing out the path of the sum of the periodic motions.

Here are some examples of the effects the program can produce. Of course, they are far more attractive and dynamic in colour on the screen than the dull old black and white printouts here suggest.





There are two versions of the program. The first one allows you to enter your own choice of STEP and X and Y frequencies. STEPs should be around 50 or more if the X and Y frequencies are in single figures, and correspondingly more as the size of the frequency is increased. The higher the STEP number, the greater the resolution of the final figure.

For a start, enter this program into your VZ300:

```
10 REM LISSAJOUS FIGURES
20 PI=3.141592
25 CLS
30 INPUT "STEPS";S
34 REM ENTER 0 TO END
35 IF S=0 THEN END
40 INPUT "Y FREQUENCY";Y
```



```
50 INPUT "X FREQUENCY";X
80 MODE (1)
82 Z=INT(RND(0)*3)+2
85 COLOR (Z)
90 FOR A=0 TO 2*PI STEP PI/S
100 SET ((20*SIN(A*Y)+60),(20*COS(A*X)+30))
110 NEXT A
120 FOR T=1 TO 2000:NEXT T
130 RUN
```

Try the following sample values with this program:

STEP	Y	X
50	1	1
60	2	1
60	3	4
60	5	2
80	1	2.5
600	13	26.5
300	7	4
175	9	4
500	13	6
500	4	12
200	8	1
250	3	8
500	20	8
500	15	20

DOING IT AT RANDOM

The next program chooses the two frequencies at random, with a number between 1 and 20. The STEP size is fixed at 500, but there is

no reason why you should not change that if you wish. There is a slight pause at the end of each figure, and then the screen clears and a new figure begins:

```
10 REM AUTO-LISSAJOUS
20 PI=3.141592
25 CLS
30 S=500
40 Y=RND(20)
50 X=RND(20)
80 MODE (1)
82 Z=INT(RND(0)*3)+2
85 COLOR (Z)
90 FOR A=0 TO 2*PI STEP PI/S
100 SET ((20*SIN(A*Y)+60),(20*COS(A*X)+30))
105 IF INKEY$<>"" THEN 40
110 NEXT A
120 FOR T=1 TO 2000:NEXT T
130 GOTO 40
```

The final version of the program is simply a modification of the preceding one. In this version, the screen does not clear between each image, so eventually you will end up with a very complex overlay of designs:

```
10 REM AUTO-LISSAJOUS
20 PI=3.141592
25 MODE(1)
30 S=500
40 Y=RND(20)
50 X=RND(20)
```



```
82 Z=INT(RND(0)*3)+2
85 COLOR (Z)
90 FOR A=0 TO 2*PI STEP PI/S
100 SET ((20*SIN(A*Y)+60),(20*COS(A*X)+30))
105 IF INKEY$<>"" THEN 40
110 NEXT A
120 FOR T=1 TO 2000:NEXT T
130 GOTO 40
```

WEAVING MARTIAN LACE

As you know, the SET command is used to place dots of colour on the screen at positions specified by the numbers which follow the command.

SET(X,Y) places a pixel at location X (which can be zero to 127, the distance across the screen from left to right) and location Y (a number from zero to 63, being the count down the screen).

Our next program, MARTIAN LACE, uses SET to produce a balanced, and highly attractive, design on the screen. As you'll see when you run it, the design continues to unfold for as long as you watch it:

```
10 REM MARTIAN LACE
20 MODE(1)
30 X=RND(63)
40 Y=RND(31)
50 Z=RND(4)
60 COLOR(Z)
```

```
70 SET(X,Y)
80 SET(128-X,Y)
90 SET(128-X,64-Y)
100 SET(X,64-Y)
110 GOTO 30
```

THE MOTION PICTURE MAN

You can use the computer as a kind of 'etch-a-sketch' toy in mode 1. You enter a number (2, 3 or 4) to select the colour you want, then use the arrowed keys to draw designs of your choice on the screen. The image begins in the top left hand corner of the screen.

```
10 REM MODE 1 SKETCHER
20 CLS:INPUT A
30 MODE(1)
40 COLOR A
50 X=0:Y=0
60 REM — DRAW PICTURE —
70 C$=INKEY$
80 IF C$="," AND X<127 THEN X=X+1
90 IF C$="M" AND X>0 THEN X=X-1
100 IF C$="." AND Y>0 THEN Y=Y-1
110 IF C$=" " AND Y<63 THEN Y=Y+1
120 SET(X,Y)
130 IF C$="C" THEN COPY
140 IF C$="R" THEN RUN
150 IF C$="N" THEN 50
160 IF C$="E" THEN 190
170 IF C$="S" THEN END
180 GOTO 70
190 GOTO 190
```


As well as responding to the arrow keys, the program accepts, and acts on, the following commands:

C - copies pictures created to some printers (such as the Seikosha GP-100 or GP-100A)

R - runs the program again from the beginning

N - moves the cursor back to the start position

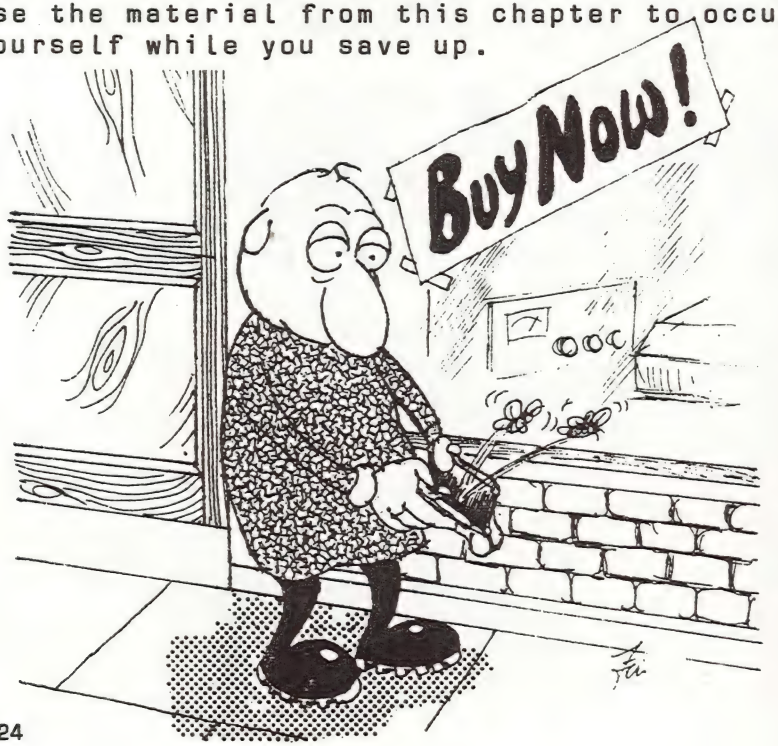
E - freezes the picture

S - stops the program

That brings us to the end of this chapter on using the graphics. You're sure to be work on the ideas given here to exploit the graphics potential of your VZ to the full.

Chapter Thirteen — Animation

There is little doubt that moving graphics games, such as the ones you see in arcades, are among the most exciting things you can run on a computer. Although the programs we select in this section of the book are in no way as sophisticated as arcade games, techniques given will allow you to write moving graphics games of your own, and will give you some insight into arcade-standard programs. So even if you're a bit short of cash at the moment to buy software, you can use the material from this chapter to occupy yourself while you save up.



THE SECRET OF ANIMATION

It is simple to make something appear to move on the VZ screen. You put a shape, such as the letter "A", on the screen in a particular spot, and wait a moment or two. Then you print a blank space where the "A" was, and at the same time, print a new "A" a short distance away from its original position. The eye will be fooled into thinking the A has moved from the first position to the second.

This process is repeated over and over again, and it creates the impression that the object is moving. That's all there is to it. Put one moving object under your control, and another one under the VZ300's control, and you have the raw ingredients of a game. Let's see how it works in practice.

USING PRINT@

As you know, in the text mode the VZ's screen is 32 characters across, and 16 lines down. The PRINT@ command works by counting the very first position on the screen (the one in the top, left hand corner) as number 0. It counts across the line, then gets to position 31 at the right hand end of the line. Position 32 is the first one at the left hand end of the second line. This goes right through the screen, with the final position, in the bottom right hand corner, as position number 511.

We'll start by putting something on the screen. Enter our next brief program and run it. When you see the question mark appear, enter two numbers separated by a comma. The first is the number of lines down you want the object to appear, and the second is the number of spaces you want the object to be across the screen. The first number must be between 0 and 15 and the second number between 0 and 31.

Enter the program, run it for a while, then return to the book:

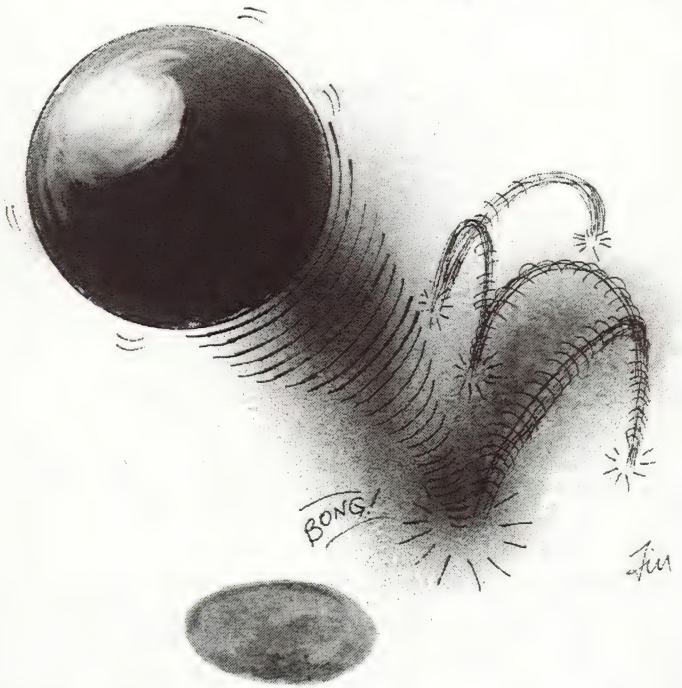
```
10 REM PRINT @ DEMO
20 CLS
30 INPUT A,B
40 PRINT @(32*A+B),"A"
50 PRINT @0,"          "
60 PRINT @0,;
70 GOTO 30
```

Note the use of the formula in line 40, which changes the numbers which have been entered as down and across co-ordinates into a single number which the PRINT@ command can use. In the next chapter, we'll use a similar formula for PEEK and POKE, but we'll stick with PRINT@ for the time being.

You'll find you're using the formula PRINT@(32*A+B), or a similar one to it, time and again in programs,

FOLLOW THE BOUNCING BALL

We'll now use PRINT@, and our formula, to create a bouncing ball program. NEW the VZ to get rid of the program which is now in



it, and enter the following program, then return to the book for a discussion on it.

You'll find the techniques that are used for animating this ball can be used in many, many moving graphics games which you write. So it is worth taking the trouble at this point to understand exactly how it works, so

you can apply the techniques to your own programs.

```
10 REM BOUNCING BALL 1
15 CLS
20 A=6:B=11
30 Y=1:X=1
40 EA=A:EB=B
60 PRINT @ (32*B+A),"O"
70 B=B+X
80 A=A+Y
90 IF A<2 THEN Y=-Y
100 IF A>30 THEN Y=-Y
110 IF B<2 THEN X=-X
120 IF B>14 THEN X=-X
130 GOTO 40
```

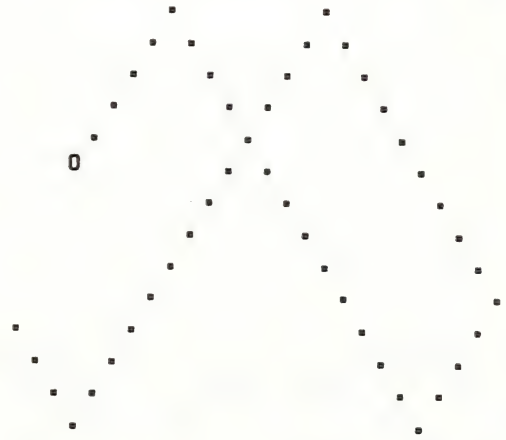
When you run this, you'll see the ball appear on the screen, bouncing from the sides, and leaving a trail behind it. I deliberately did not erase the 'old' ball in this program, so you could see that it really was just a series of the same character being printed at different positions on the screen.

Now, add this line and see what happens:

```
125 PRINT @(32*EB + EA),"."
```

When you run the program again, you'll see

something like this on the screen:



As you can see, it leaves a trail of dots showing where the ball has been. Remove the dot from between the quote marks in line 125, and run the program again, to see a true bouncing ball.

This program gives a very good impression of a moving object. As I said earlier, this simple bouncing ball demonstration shows a number of key things about moving graphics programs, so we'll go through the program line by line, explaining what each line is doing.

15 - clears the screen

20 - sets the start position of the ball, at 7 across and 12 down

30 - sets X and Y which control the change in position of the ball between each movement

40 - sets two variables used to erase the old position. These are called EA (for 'erase A') and EB (for 'erase B'). It's not a bad idea to use variable names which suggest the job the variable is doing in the program, as it makes it much simpler later on when you're going through the program, to keep track of what each bit of it does. Note, and this is important, EA and EB are set equal to A and B just before the ball is printed at 32*B+A (line 60) and before A and B are modified (lines 70 and 80)

60 - this prints the ball on the screen

70 - B (the down position) is changed

80 - A (the across position) is changed

90 - this checks that the A position is not too far to the left, and if it is, changes Y to minus Y, which causes the ball, next time it comes to this line, to reverse its direction, so it appears to bounce

100 - does the same for the right hand side

110 - checks the up/down position, and if the ball is too close to the top of

the screen, changes X to minus X which, as you'd guess, causes the ball to bounce down from the top of the screen

120 - does the same for the bottom of the screen

125 - prints a blank on the 'old' position of the ball (that is, the position designated by A and B before they were modified by lines 70 and 80)

130 - sends action back to line 40, where EA and EB are set equal to the ball's new position, before it is reprinted in the new position with line 60

Although this explanation is fairly long, and much longer than the program it is explaining, it is worth reading through it carefully. Once you understand it, you'll be well on the way to writing your own programs.

Now modify the program so it is as follows. In this version, there is a little object at the bottom of the screen which is your 'bat' and you have to get the ball to bounce off it to keep the ball moving.

You use the 'Z' key to move left, and the 'M' key to move the bat at the bottom of the screen to the right:

```
10 REM BOUNCING BALL 2
15 CLS
20 A=6:B=11:F=11
30 Y=1:X=1
40 EA=A:EB=B
50 PRINT @(445+F),"  ";CHR$(131);"  "
60 PRINT @ (32*B+A),"O"
70 B=B+X
80 A=A+Y
82 A$=INKEY$
85 IF A$="Z" AND F>3 THEN F=F-2
86 IF A$="M" AND F<29 THEN F=F+2
90 IF A<2 THEN Y=-Y
100 IF A>30 THEN Y=-Y
110 IF B<2 THEN X=-X
120 IF B>12 AND ABS(F-A)<3 THEN X=-X
121 IF B=16 THEN END
125 PRINT @(32*EB + EA)," "
130 GOTO 40
```

Now, this is not a particularly satisfying game (it probably doesn't even deserve the title 'game'), but it shows some extra ingredients you'll probably be using yourself in moving graphics games. In this program, outside interaction is brought into play for the first time (you moving the bat) and the computer responds to this (keeping the game underway if the bat is in the right position).

Here's what the extra lines do:

20 - F is the position of the bat across the screen

50 - this prints the 'bat'. If you prefer, instead of using CHR$(131), you can use the graphics character available directly from the 'Y' key (using SHIFT). Note that there must be two blank spaces within the quote marks on both sides of the character, to 'un-print' the bat as it moves

82 - this read the keyboard, with INKEY$, and sets the result of this reading equal to A$

85 - if A$ equals "Z" (that is, if you are pressing the "Z" key) and F is greater than 3, the value of F is reduced by 2. As F controls the position of the bat across the screen, reducing F moves it to the left

120 - if B is greater than 13 (which means the ball is near the bottom of the screen), the position of the bat is checked, and if it is close to the ball, a bounce occurs, and the game continues

121 - if the ball has missed the bat, this line halts the program

This is not, as I said, a very inspiring or challenging game, but with a few minor changes can be given a bit of value. Modify

it so that it reads as follows, and play with it a bit:

```
10 REM BOUNCING BALL 3
12 REM 'SQUASH'
15 CLS
17 PRINT @ 227,"————————————————"
18 FOR J=256 TO 416 STEP 32
19 PRINT @J,"  >                <":NEXT
20 A=6:B=11:F=11
25 SC=1
30 Y=1:X=1
40 EA=A:EB=B
50 PRINT @(445+F)," ";CHR$(131);" "
60 PRINT @ (32*B+A),"*"
70 B=B+X
80 A=A+Y
82 A$=INKEY$
85 IF A$="Z" AND F>8 THEN F=F-2
86 IF A$="M" AND F<24 THEN F=F+2
90 IF A<7 THEN Y=-Y
100 IF A>17 THEN Y=-Y
110 IF B<9 THEN X=-X
120 IF B=13 AND ABS(F-A)<3 THEN X=-X
121 IF B=15 THEN END
122 IF B=13 THEN SC=SC+367:PRINT @68,"SCORE IS ";SC
125 PRINT @(32*EB + EA)," "
130 GOTO 40
```

Again you'll have to use the 'Z' and 'M' keys, but this time you have a smaller 'playing field', and there is a bit more of a challenge to the game. You might like to spend some time playing with this program, modifying it as you see fit, before returning to the book to discuss moving an

object up and down, as well as to the right and left, on the screen.

TRAPPER

We now have a program which is somewhat more difficult – and enjoyable – to play, in which an 'X' is stalked by the prehistoric '@' symbol. Stirring stuff.

In this program, you are the 'X' and the VZ300 controls the '@'. You use the following keys to move yourself around the

screen, trying to stay out of the VZ's clutches for as long as possible:

A - to move up
Z - to move down
M - to move left (see arrow on this key)
, - to move right (see arrow)

```
10 REM TRAPPER
15 CLS:T=0
20 BA=5:BD=5
30 TH=20:TV=14
40 EA=BA:ED=BD
50 EH=TH:EV=TV
60 PRINT @ (32*ED+EA)," "
70 PRINT @ (32*TV+TH)," "
80 A$=INKEY$
90 IF A$="A" AND BD>2 THEN BD=BD-1
100 IF A$="Z" AND BD<14 THEN BD=BD+1
110 IF A$="," AND BA<30 THEN BA=BA+1
120 IF A$="M" AND BA>2 THEN BA=BA-1
130 PRINT @ (32*BD+BA),"X"
135 IF RND(2)=2 THEN 150
140 IF BD>TV THEN TV=TV+1
145 IF RND(2)=2 THEN 160
150 IF BD<TV THEN TV=TV-1
155 IF RND(2)=2 THEN 170
160 IF BA<TH THEN TH=TH-1
170 IF BA>TH THEN TH=TH+1
180 PRINT @ (32*TV+TH),"@"
185 T=T+1
190 PRINT @ 0,"TIME ELAPSED"T
210 IF BD=TV AND BA=TH THEN SOUND RND(20),1:GOTO 210
220 GOTO 40
```


There is no need to go through this program line by line, as I'm sure that by now you have a pretty good idea of what effect the various parts of the program have.

The main differences between TRAPPER and the BOUNCING BALL series of programs is that we are now moving the object under our control in two dimensions, and that the thing which is trying to trap us, can 'sense' (using lines 135 to 170) just where we are. The lines 135, 145 and 155 are just to give you a chance. Take them out, and you'll be dead within ten moves, every single time.

This program points up one of the disadvantages of working in BASIC, especially with PRINT@. It is fairly slow, and everything we add to a program slows it down further.

Fortunately, there is another way to move things on the screen, using PEEK and POKE. We'll be looking at those in the next chapter.

Chapter Fourteen — PEEK and POKE

PEEK and POKE are weird words. Unlike GOTO, PRINT and IF/THEN, they are not the sort of words which immediately suggest what they're used for.

In fact, PEEK and POKE seem to inspire more fear and confusion in programs when they first come across them than any other words in BASIC. There is no need to worry. They are simple to use, and their use follows the ways we used PRINT@ in chapter 13.

Although they can be used in programs in place of PRINT@, PEEK and POKE are not just

strange-sounding substitutions for them. There are two advantages in using them. Firstly, POKE is faster than PRINT@. Secondly, PEEK can be used to find out what is on the screen at a particular position. This is ideal for discovering whether or not an alien has been shot, or the plane you are flying has hit the side of a mountain.

When you POKE something onto the screen, you put it there. When you PEEK the screen, you are looking to see what is on the screen at that particular position. And that's just about all you have to remember.

If you have two co-ordinates of a position, such as A (for 'across') and D (for 'down), you PRINT@ them, as explained in the previous chapter, using the formula PRINT@(32*D+A). To POKE something onto the screen, you use a line which is related to it, as follows:

```
POKE 28672+32*D + A, X
```

I generally use 28672, the location of the top left hand corner of the screen, or 28736, the position two lines below it. The X is the character number, and this is the character which will appear when we POKE directly to the screen.

You'll remember that, when working with PRINT@, we used a space (" ") to wipe something off the screen. The POKE equivalent to this is 32, so...

```
POKE 28672 + 32*D + A, 32
```

...is equivalent to...

```
PRINT @ (32*D + A)," "
```

If you want to find out what is at a particular address (the locations on the screen are called 'addresses'), you use PEEK as follows:

```
IF PEEK (28672 + 32*D + A) = ... THEN ...
```

Now, PEEK and POKE are simpler to use than they may appear at first sight. I suggest you enter the following programs, which use the two commands, and follow through the listings carefully. This will allow you to learn about the use of the two words more quickly than any other way.

MESOZOIC ATTACK

The Mesozoic Era was the time when fierce dinosoars, like the Dickus Smithus Electronicus, roamed the earth. Our program, MESOZOIC ATTACK, is really a PEEK/POKE version of TRAPPER. Only POKE is used in this program, to introduce you to the new words in a gentle way.

The control keys are the same as in TRAPPER, with 'A' and 'Z' to move you up and down, and the arrow keys 'M' and ',' to move you left and right.

```
10 REM MESOZOIC ATTACK
15 HS=0
20 COLOR ,0:CLS
30 SC=0
40 HA=1:HD=1:D=15:A=30
50 EA=HA:ED=HD
60 IF INKEY$="A" THEN HA=HA-1
70 IF INKEY$="Z" THEN HA=HA+1
80 IF INKEY$="," THEN HD=HD+1
90 IF INKEY$="M" THEN HD=HD-1
91 IF HA<1 THEN HA=1
92 IF HA>14 THEN HA=14
93 IF HD<1 THEN HD=1
94 IF HD>31 THEN HD=31
95 EB=SB
100 SB=28672 + 32*D+A
110 IF SB=H THEN 1000
115 POKE EB,143
120 POKE SB,(36+RND(2))
130 H=28672 + 32*HA+HD
140 IF SB=H THEN 1000
145 POKE EH,143
150 POKE H,88:EH=H
160 IF D<HA THEN D=D+1
165 IF RND(9)>3 THEN 180
170 IF D>HA THEN D=D-1
175 IF RND(9)>3 THEN 190
180 IF A<HD THEN A=A+1
190 IF A>HD THEN A=A-1
```

```
200 SC=SC+1
210 PRINT @ 20,"TIME>"SC
240 IF D<1 THEN D=1
250 IF D>14 THEN D=14
260 IF A<1 THEN A=1
270 IF A>31 THEN A=31
275 IF SB=H THEN 1000
280 GOTO 50
1000 POKE SB,143:POKE H,143
1001 POKE EB,143:POKE EH,143
1002 FOR J=1 TO 19
1003 POKE SB,(80+RND(19))
1005 PRINT @ 453,"I GOT YOU!!!"
1010 SOUND RND(20)+1,1
1020 NEXT J
1030 PRINT @226,"YOU SURVIVED FOR"SC
1035 IF SC>HS THEN HS=SC
1040 PRINT @258,"BEST SO FAR IS"HS
1050 FOR T=1 TO 500:NEXT T
1060 GOTO 20
```

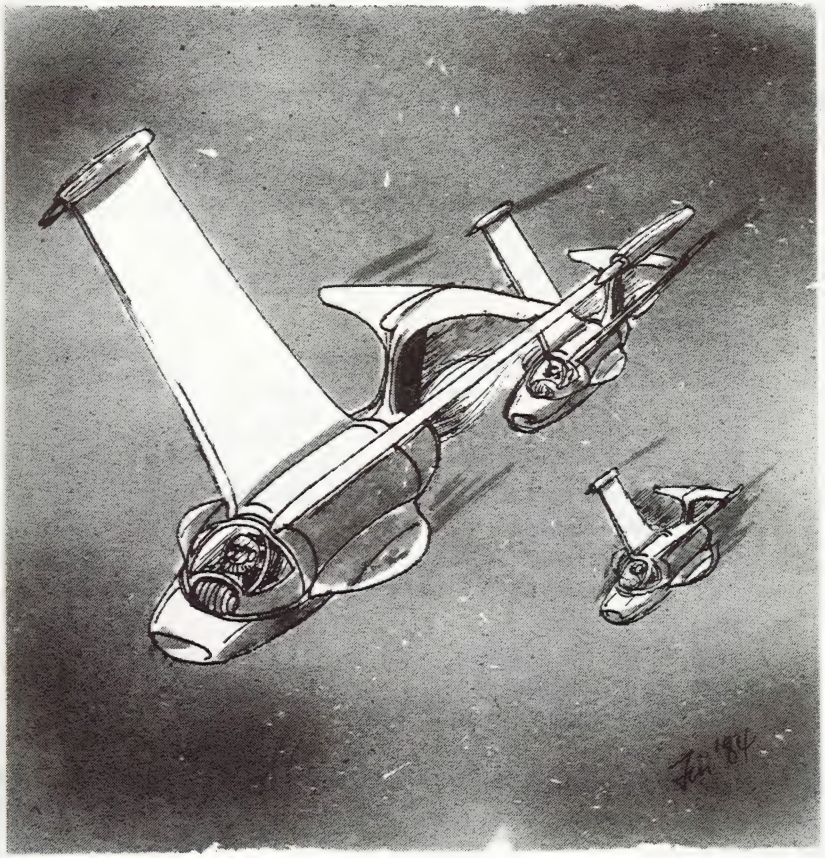
And from the distant past, we move to the far distant future with our next program, which uses PEEK to 'read' the screen.

V-WING SPACE BATTLE

In this program, you pilot your little V-wing space craft around the VZ screen, trying to run over the numbers which appear on it. Each number will appear for a limited amount of time, and if you run over it, a score related to that number will be added to your growing tally.

PEEK is used here, for the first time in one

of our programs, in line 190, where it looks at the position where your space craft is



about to be POKEd, and if it finds a number there, increments your score.

```
10 REM V-WING SPACE BATTLE
20 CLS
30 SC=0
```

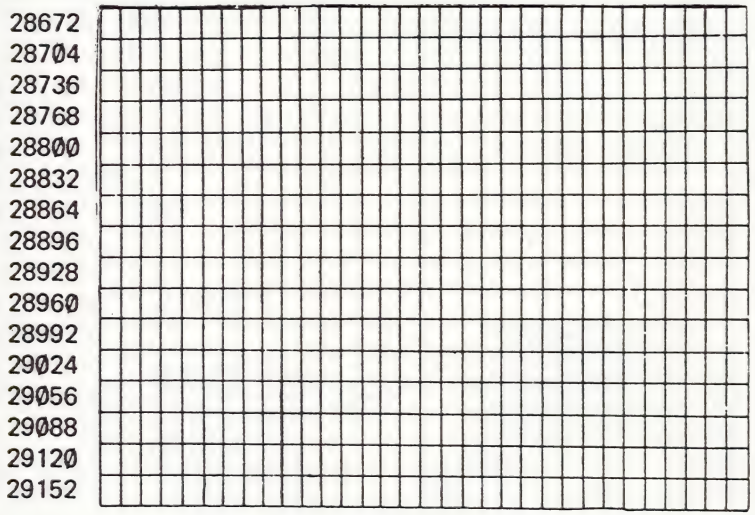
```
100 FOR Z=1 TO 20
101 COLOR,0
110 SL=28736:M=22:D=-32
120 POKE (28715+RND(467)),(RND(9)+48)
130 W=SL
140 A$=INKEY$
150 IF A$="," THEN SL=SL+1:M=62
160 IF A$="M" THEN SL=SL-1:M=60
170 IF A$="A" AND SL>28736 THEN SL=SL-32:M=1
180 IF A$="Z" AND SL<29151 THEN SL=SL+32:M=22
190 Q=PEEK(SL)
200 IF Q>48 AND Q<58 THEN SC=SC+Q:GOTO 980
205 POKE W,143
210 POKE SL,M
220 IF RND(0)<.99 THEN 130
230 COLOR,1:FOR T=1 TO 20:NEXT T:COLOR,0
980 CLS:PRINT @ 0,"SCORE  ";SC;"  ";20-Z;" SHIPS LEFT"
982 COLOR,INT(RND(0)*2)
985 SOUND RND(25),1:IF RND(10)>6 THEN 982
990 COLOR,0:NEXT Z
995 CLS:PRINT:PRINT:PRINT
1000 PRINT "THE BATTLE IS OVER       "
1003 PRINT:PRINT
1005 PRINT "YOUR SCORE IS"SC
1100 COLOR,INT(RND(0)*2)
1120 SOUND RND(25),1:GOTO 1100
```

You can see from how the screen is 'read' in line 190, and the use which is made of that information in line 200 (which adds to your score, then ends action to line 80 to prepare the next 'target'), how PEEK can greatly simplify your programs where you must discover if two objects have collided.

You should also be pleased to discover that

PEEK is much faster than checking co-ordinates (IF A=Z AND B=K THEN...). This increased speed, of course, makes for a much more satisfying game.

Here is a grid, showing the positions on the POKE screen:



Finally in this chapter, here are the character code numbers for use with PEEK and POKE:

0 - @	1 - A	2 - B	3 - C
4 - D	5 - E	6 - F	7 - G
8 - H	9 - I	10 - J	11 - K
12 - L	13 - M	14 - N	15 - O
16 - P	17 - Q	18 - R	19 - S

20 - T	21 - U	22 - V	23 - W
24 - X	25 - Y	26 - Z	27 - [
28 - \	29 -]	30 - ↑	31 - ←
32 -	33 - !	34 - "	35 - #
36 - $	37 - %	38 - &	39 - '
40 - (	41 -)	42 - *	43 - +
44 - ,	45 - -	46 - .	47 - /
48 - 0	49 - 1	50 - 2	51 - 3
52 - 4	53 - 5	54 - 6	55 - 7
56 - 8	57 - 9	58 - :	59 - ;
60 - <	61 - =	62 - >	63 - ?

Chapter Fifteen — Structured Programming

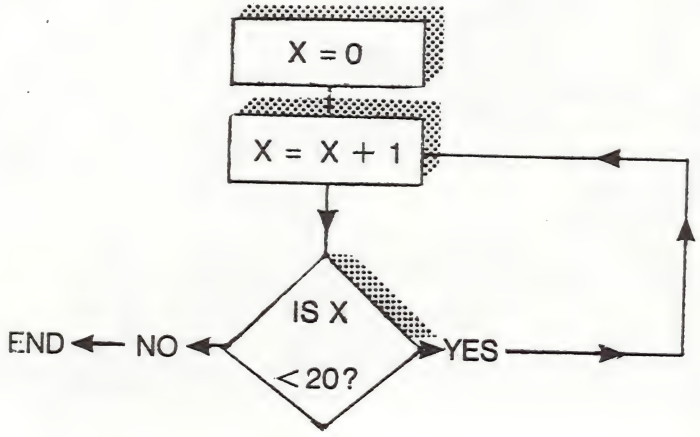
There will come a point in your development as a programmer when you'll have mastered the use of much of the use of BASIC on the VZ300, and can now concentrate on writing better programs; programs which work after relatively little debugging, which are easy for others to understand and operate, and which are written logically and elegantly.

Your programs will be more likely to run first time if they are planned out carefully before you start actually programming into the VZ300.

A good way to start is to use a diagram which is often called a 'flow chart'. A flow chart is a series of boxes and other shapes, joined by lines, which show the flow of action and decision-making within the computer while the program is running.

The shapes used are not too important, and I suggest you stick to just two: a rectangle for most actions the computer must carry out, with a diamond shape each time the computer has to make a decision. The corners of the diamond can be used - as you can see in the diagram - to cater for the alternatives facing the computer.

The diagram shows the flow chart of a program which sets the variable X equal to zero, then adds one to it. The value of X is checked. If X is found to be less than 20, the program goes back to add one to X again. This continues until the value of X becomes equal to 20.



One advantage of using a flow chart is that you do not get locked, at an early stage of your work, into the peculiarities of VZ300 BASIC, its weaknesses and strengths, which you are using. Instead, you concentrate on what you want to do. Of course, you may have to modify your plans slightly to accept limitations dictated by your programming language, whatever it may be, but you do not have to bend to these limitations (which may be, in part, imaginary) at the start of the process.

A flow chart is 'universal'. The same flow chart can be used as the basis of a program written on some other computer which is furnished with a completely different language from that which is on the VZ300.

A flow chart models the flow of action and logic within a program, and is therefore very useful for picking up potential bugs at the earliest stages. You may, for example, find that one condition for the program will test will never be fulfilled, possibly leading to the program being trapped in an infinite loop. Other parts of the code may be bypassed completely, because the condition which triggers entry into that part of the code will never be met.

Once you've devised a flow chart for your program, and you've run through it mentally a few times, so that the most obvious bugs are removed, you should reduce the flow

chart to a series of subroutine calls. Although it seems pretty silly to do this for a simple program like our "SET X EQUAL TO ZERO, ADD ONE, CHECK IF LESS THAN 20" program, this method comes into its own with more complex programs.

PROGRAMMING IN MODULES

I think it's a good idea to start a long and complex VZ300 program with a series of sub-routine calls, with each action of the program being looked after by a separate subroutine. Then, if the steps within a

program have to be performed several times in a particular sequence, the series of subroutine calls can be cycled through - over and over again - until a particular condition is met which signals the end of the run.

You'll recognize how useful this approach to programming can be when you get to the debugging stage of your program. If there is a bug, it is likely to be within a single subroutine, so it will be relatively easy to pin down the subroutine which contains the bug, rather than having to work right through the program trying to track it down.

Working with subroutine 'modules' in this way allows you to test sections of the program in isolation, even before the entire program is finished. I'll try to make this statement clear by showing you the first part of a typical program to play Draughts on the VZ300. Our ideal Draughts program could start like this:

```
 10 REM VZ300 DRAUGHTS
 20 GOSUB 9000: REM INITIALISE VARIABLES
 30 GOSUB 8000: REM PRINT BOARD
 40 GOSUB 7000: REM ACCEPT PLAYER MOVE
 50 GOSUB 8000: REM PRINT BOARD
 60 GOSUB 5000: REM COMPUTER MAKES MOVE
 70 IF (human has not won) AND (VZ300
          has not won) THEN 30
 80 IF (VZ300 has won) THEN PRINT "I WIN"
 90 IF (human has won) THEN PRINT "YOU WIN"
100 END
```

You could quite a bit of this program running, and tested (such as the initialisation routine, printing the board and accepting the player's move), before you even turned your attention to how on earth you were going to get the VZ300 to make its move.

You would then know - for example - that you would not need to waste any extra thought on whether or not an error in the board-printing routine was the cause of odd output. Having tested the board subroutine and the player move routine, you'd know that the error must be in the subroutine between lines 5000 and 6999, the subroutine in which the computer makes its move.

All you need to do is put a single PRINT statement, such as "THIS IS VZ300 MAKING A MOVE" followed by a RETURN for incomplete subroutines, knowing that the program will accept that, and demonstrate the direction the program flow is following, even if whole sections of code have not yet been written.

In general, I'd advise you to use this system of using a 'master loop' of subroutine calls within which you will 'hold' the entire program.

I suggest you try and do as much writing of the program as possible before you turn the computer on, even though there is a great temptation to dive straight onto the VZ and

start punching in code. You'll find that the discipline of writing it out by hand in advance will serve you in good stead and should, in the long run, produce a better program than might otherwise have been the case.

Overall, you'll probably end up spending up less time on the program working in this way than you would if you began the process sitting at the VZ300 keyboard.

It took me a while to learn this lesson. Although I had read suggestions along the lines of 'work out exactly what you're going to enter before you start at the computer' in several books, I tended to just jump right in without much prior thought.

Although I worked out rough flow charts, and had an idea what sort of display organisation I wanted, I certainly did not write much program out on paper before starting at the computer. Then, I once found myself stuck for a two-week period without a computer and ideas for several programs just itching to be written. I had to write them out in an exercise book.

The relative ease with which the programs were debugged when they were eventually entered into the computer, and the complexity of the programs I wrote in this way (including my first Chess program) convinced me that this was the way I would

work from then on. It is amazing how much cleaner a program can be if all the rough working out is done on paper, rather than on the computer screen.

When working on programs which threaten to be big and involved, I tend to write the major 'call subroutines loop' first of all, but without line numbers, so the program contains lines like GOSUB PRINT BOARD and GOSUB INITIALISE. Next, I write each module (or subroutine) on a separate sheet of paper. Then, when the major subroutines have been written I shuffle them into an order which seems most logical.

All this, of course, occurs before any line numbers are written in. The subroutine modules are put in an order which ensures that the program structure is as logical as possible. I use arrows to indicate the destination of GOTO's within a module, and names for subroutine, as suggestion in the major loop. Later on, when the program has started to assume a firm shape, the lines are numbered (I usually work in tens, starting at line 10) and the relevant GOTO and GOSUB destinations are added.

All programs have an 'end condition' at which point computation stops. It is worth putting a test for this end condition as part of the GOTO which sends the program back to start cycling through the major subroutine loop. This ensures that the cycle

will continue until a particular condition is met, at which point the program 'falls through' that GOTO and continues on to the lines which signal the end of the program.

INPUT PROMPTS

It is very useful, when writing a program, to keep in mind how the program will appear to a stranger when it is run by him or her for the first time. If there is an input prompt required, it is far more useful if the program prints up something like "HOW MANY HOURS HAS THE EMPLOYEE WORKED THIS WEEK?" instead of just "INSERT HOURS?" or the almost useless lone question mark.

The same suggestion applies to print output. It is far better that your program is written so that it prints out THE NUMBER OF HOURS WORKED ON FULL PAY THIS WEEK IS 27, rather than HOURS, FULL: 27 or an unsupported 27. Of course, providing explicit input prompts and output PRINT statements consumes memory as well as typing time when entering the program, but the contribution they make to the final program means the trouble involved is well worth it.

While exact PRINT statements and input prompts will help a person running a program make sense of it, REM statements can help make the program clear to those who are examining the listing for the first time.

REM statements (which are, as you know, ignored by your VZ300 when a program is run) should be used to help illuminate the flow of logic within the program, and what is happening in certain places within it. This is especially important in parts of the program where decisions are made, or calculations are carried out.

Not only can REM statements be used to explain different parts of the code, but they can also be used to provide visual 'breaks', so that the various blocks of code which carry out certain tasks are visually separated from the rest of the program. A blank REM statement (the word REM standing alone in a program line) can be used for this. A row of asterisks is an effective alternative.

VARIABLES

It is worth considering the use of explicit names for variables, using either the word in full (such as HOURS as a variable name in a payroll program for hours worked) or an abbreviated version of this (such as HR) which has a fairly obvious meaning. You'll discover that this makes it easy, when working on a program, to keep track of your variables. This will help you with the initial debugging and later on as well if you need to improve or extend the original program.

Explicit variable names also help make your code more 'transparent' so other programmers can work out what the various parts of the program are intended to do. You'll also find it a great help to yourself when you return to the program at a later date. It is surprising how code which seemed incredibly clear it terms of its purpose when you entered it, becomes exceedingly hard to follow when you return to it after a long break.

CHECKING INPUT

Any input entered by the user should be checked by the program before it is accepted to ensure that incorrect data does not cause the program to crash at some future point. Whether you want string, or numeric input, it is often wise to allow for string input, which is first checked to see whether the entered material is acceptable and then, if necessary, the string can be changed into a number.

For example, if the user needs to enter a number between one and nine, a string can be accepted and then checked to ensure that it is not less than "1" or more than "9" before being changed into a number with a command like VAL.

As well as ensuring that the program will reject invalid input, you should check the program to see all the inputs which it does

accept produce sensible answers when processed later on in the program. For example, make sure that your program does not accept zero as a possible number if the trusty VZ must later divide by this number.

Similarly, if numbers are to be processed by a function, and then the result of this processing used for division, you must check that an apparently valid input does not turn into zero as a result of evaluation by the function.

If the information entered by the user is rejected, and a new input is requested, the program should ideally point out why the original input was not suitable, or spell out again exactly what is required (such as "ENTER A NUMBER BETWEEN 1 AND 4"). You risk making users angry if input which appears valid is continually rejected without apparent reason.

DOCUMENTATION

The written material which accompanies a program is often called documentation. If you're preparing programs to be used by other people, it is very useful for those programs to be supported by some documentation, however sketchy.

The written information should explain, of

course, what the program does, then go on to outline the flow of action within the program. The documentation should alert the user as to the kind of actions which will be required from him or her when running the program, and give an indication of the kinds of user input and reaction which will be accepted.

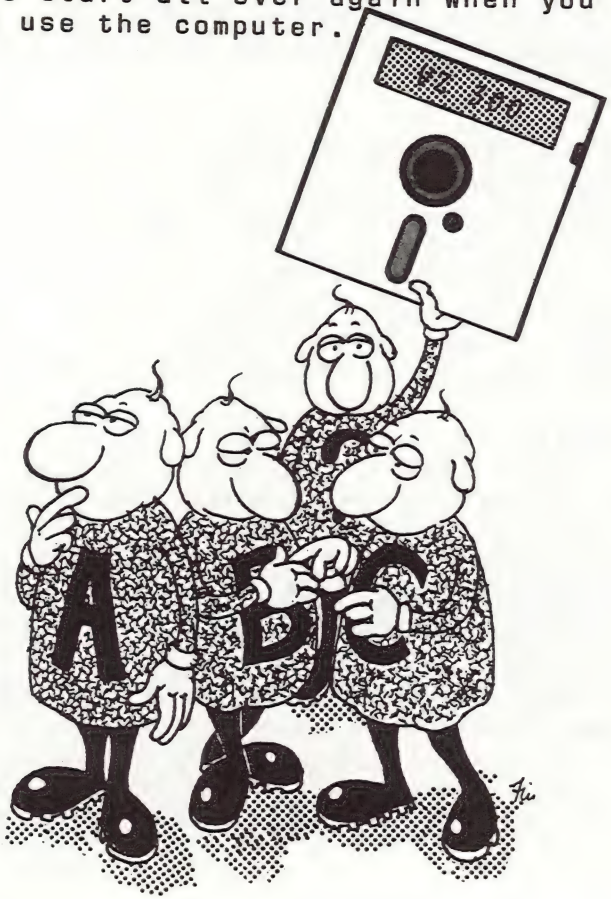
The format of the final output should also be discussed. A list of variable names can be included.

If there are ways in which the program can be developed, extended or improved, suggestions along these lines can be added to the documentation. Written references to any material which will help in understanding the algorithms used, or for giving suggested areas for program development, should also be included.

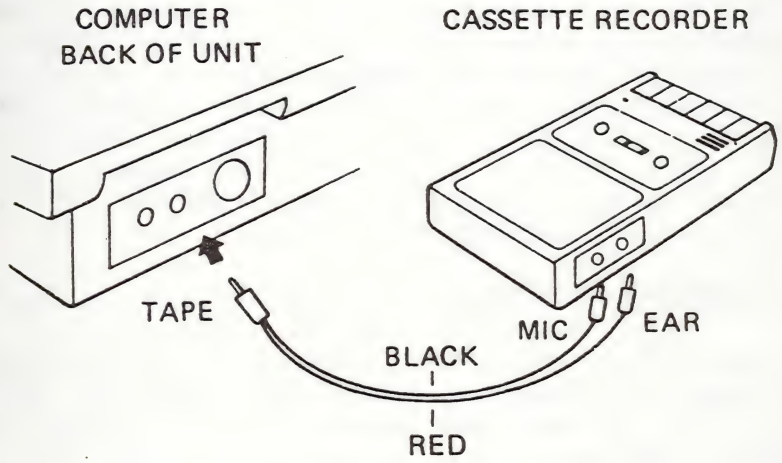
Documentation for a major program should start with an introduction which quickly explains what is going on, and tells how to use the program. The later parts of the documentation can then discuss the program in greater detail. It is not good practice to force the user to wade through a vast amount of information in order to dig out the vital facts he or she needs to get the program running.

Chapter Sixteen — Cassettes and Disks

It is really rather pointless writing great programs, or typing in programs from books like this one, and then not saving them on cassette or disk before you turn your VZ300 off. If you don't save the program, it is lost when you turn off the power, and you have to start all over again when you next want to use the computer.



To connect up a cassette player, using either your own cassette, or the special one for the VZ provided by Dick Smith Electronics, use the cords which come with your computer. Connect them up as shown in the diagram.



To save a program on cassette, so you can use it again, you have to give the program a name. This name can be from one to 16 characters long, so MYPROG, A4, Z and SAUSAGESANDSAUCE are all valid names. The important thing is that the file name must start with a letter, rather than a number or symbol.

SAVING A PROGRAM

You put the cassette in your recorder, make sure the tape is past the blank leader

portion, then type in on your VZ:

CSAVE "NAME"

Of course, you put your own name for the program in place of the word NAME. Then, start the cassette player, in record mode, and once the tape is rolling, press the RETURN key on your VZ. The flashing cursor will vanish while the save takes place. Once the cursor re-appears, the program has been saved on cassette. It is suggested that you save each program two or three times, just to make sure you have a version of it which will load successfully next time.

VERIFYING A PROGRAM

To check the program on the tape, before you NEW your VZ300, rewind the tape to the position it had just before the recording began. Type in:

VERIFY "NAME"

Then, press the PLAY button on your recorder, and the RETURN key on the VZ. The computer will compare the program on tape with the one in your computer's memory, and if they are the same, will print up the message VERIFY OK. This ensures that you have an uncorrupted version of the program on tape.

LOADING A PROGRAM

To get a program from tape back into your VZ300, set the tape up so it is just before the start of the program. (To make it easier to do this, you might like to precede the saving of a program with a short voice announcement, so you know which program is coming next. If you do, make sure you disconnect the microphone before saving the program.)

Then, type in:

CLOAD "NAME"

Now press the play button on your recorder and the RETURN key on the computer.

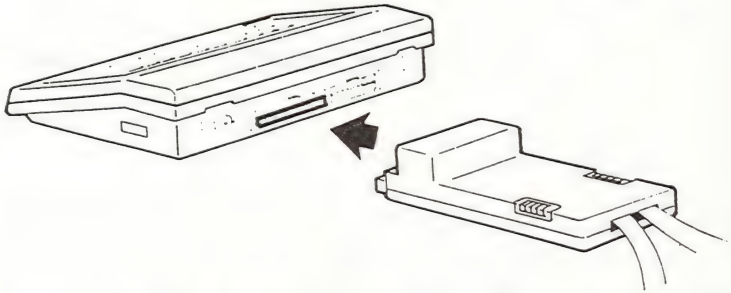
The message WAITING will appear on the screen while the computer waits for your program on the tape. When it appears, you'll see the words FOUND T: NAME as other programs are located. The message LOADING T: NAME means the program name you have specified has been located, and is now loading into the VZ300.

ATTACHING A DISK DRIVE

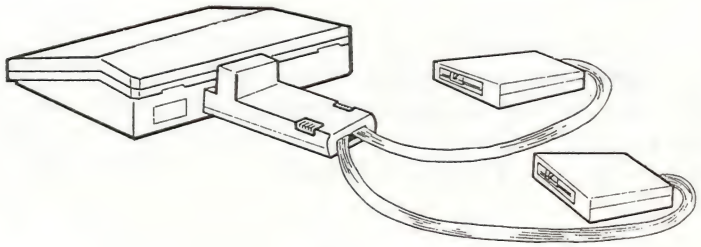
While cassettes have the advantage of being very cheap as a means of storing programs, they are slow to load and save, and somewhat unreliable. Disks, by contrast, are not so cheap, but load virtually instantly, and

with close to 100% reliability.

To connect the X-7302 disk drive to your VZ300, you need the floppy disk controller (X-7304) and a 16K memory expansion module (X-7306). You start by turning the power off to your computer, then sliding the disk controller cartridge into the slot at the back of the computer. Next, you plug your 16K (or 64K) memory expansion unit into the slot at the top of the disk controller.



Then you plug the connector on the flat cable provided with the disk drive into the controller socket marked DRIVE 1, connect the separate power supply to the disk drive, and then turn on the power to the disk drive, and the power to your VZ300.



If you have a second disk drive, you plug it into the socket marked DRIVE 2, before turning on the power.

Once you do turn on the power, the message on the screen will read:

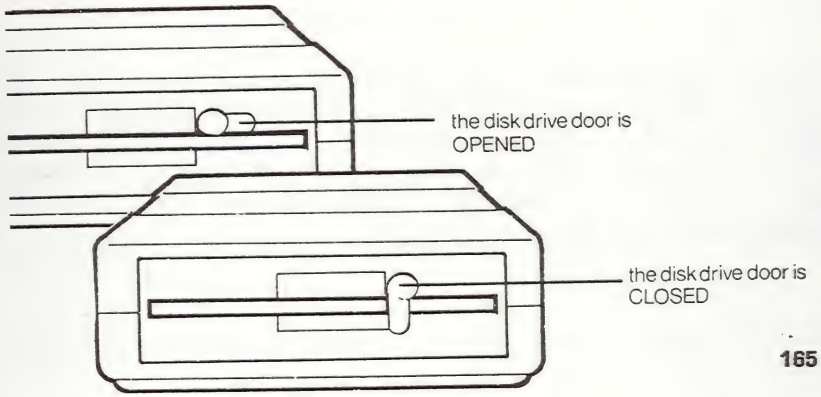
```
VIDEO TECHNOLOGY
DOS BASIC V1.2

READY
```

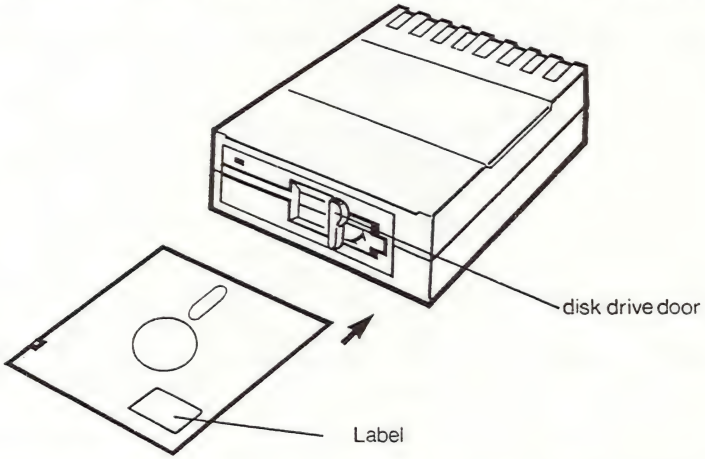
If you do not get this message, turn off power to all units, and go through the procedure again, making sure you are always gentle when sliding parts into each other.

USING DISKS

Once you have the system up and running, you can experiment by saving a program. Open the disk drive door, by first turning the little handle upwards to make room for the disk, and after you've inserted the disk, turn the little handle down again.



You need to format a disk before it can be used. To do this, get a blank disk, or one which you no longer need (because formatting



wipes everything off a disk), and place it in your disk drive.

Now type in:

INIT

This stands for 'initialise'. Once you do this, and press RETURN, the red light will come on and you'll be able to hear the disk spinning around inside the drive. Never, ever try to remove a disk from the drive when the red light is on.

SAVING PROGRAMS TO DISK

Once you have an initialised disk, you can use it to save a program on.

Whereas with cassettes you used the command CSAVE "NAME", you leave the 'C' off, and simply enter:

SAVE "NAME"

The red light will come on, the drive will make a few sounds for a second or two, and then the word READY will appear to show you the process is complete. Note that, unlike when you work with cassettes and files can be up to 16 characters long, the file name for a disk file cannot be more than eight characters long.

GETTING THE PROGRAM BACK

It is very simple to load in a program from disk. You just type in:

LOAD "NAME"

After a few seconds of sound from the drive as the red light comes on, the word READY will appear on the screen to show you that the program has loaded. If you have only ever used cassettes you'll be extremely impressed by the incredible speed with which disks work.

Then, to run a program, you just type in RUN

as normal, and the program will run. If you want to avoid this step, then, instead of LOAD "NAME" you type in:

RUN "NAME"

This will ensure that the program runs automatically, once it has loaded.

WHAT'S ON THE DISK?

The word DIR (which is short for 'directory' is used to discover which programs you have on your disk. When I typed in DIR, and then pressed RETURN with one of the disks I developed when writing this book, I saw the following on the screen:

```
T:TEMP
T:TEMP1
T:GOLF
T:REACTION
T:SPIRO
T:LISSA-1
T:LISSA-2
T:MARTIAN
T:SKETCH
T:TRAPPER
T:MEZOIC
T:VWING
T:LIFE
```

To find out how much space you have left on your disk, you enter the word STATUS. When I did it with my program disk still in place,

the following message came up on my screen:

```
STATUS
537 RECORDS FREE
67.125K BYTES FREE
```

OTHER DISK COMMANDS

The other two commands you're going to need when working with disks are as follows:

REN

REN is used to rename a file on disk. To use it, you type in:

REN "OLDNAME","NEWNAME"

Then, when you next do a DIR, you'll see that the file is not listed under its original name, but now appears in the directory under the new name.

ERA

You use this to wipe an unwanted file from your disk. As it is impossible to save a file of the same name as another one on the disk, you may want to erase an earlier version of the program to make room for the newer one. To use, just enter:

ERA "NAME"

Your next DIR will show that the file has been wiped.

TAKING CARE OF DISKS

There are a few commonsense rules for looking after the disks you use with your VZ300 which will ensure you get the maximum value out of them. It will also reduce the chance of losing valuable data through damage to a disk.

if anything happens to the disk - such as it getting bent or having something spilled on the outer cover (and not into the disk itself), repair the damage as best you can, and make a copy of the disk immediately. Then discard the original disk.

If you put a contaminated disk in your drive, you run the risk of damaging the read/write head.

Don't touch the part of the disk which shows through the slot in the outer cover, and don't put anything (such as cups of coffee, ash trays, copies of the Dick Smith catalogue or other heavy books) on top of either the disk, or the cardboard cover.

THE NASTY MAGNETIC FORCE

Magnetism can corrupt data on a disk. If you have anything which is, or could be, magnetised, such as radio speaker, a pair of

scissors, or a telephone handset, keep it well away from your disks.

Disks, as I guessed you've realized by now, are pretty sensitive creatures. They don't like heat too much, so leaving them in the car while you go surfing is going to cause a few problems. Disks work best in the range 10 to 51 degrees Celsius. If a disk is exposed to temperatures outside this range, leave it for at least half an hour to adjust to room temperature before putting it in the disk drive.

You should never bend disks, or use rubber bands, adhesive tape or paper clips on them.

If you want to write on the cardboard outer sleeve, take the disk out first. Write on the label, which you are going to stick on the disk itself, before you stick it in place. If you have to add something later, write gently, and with a felt-tipped pen. Don't use a biro or sharp pencil.

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES

TIM HARTNELL TAKES THE BYTE OUT OF PROGRAMMING YOUR VZ300!

Now you can learn to program your Dick Smith VZ300 Personal Colour Computer in a matter of hours. Tim Hartnell leads you step-by-step in this complete guide for the first-time programmer (plus there's quite a bit of material for those with previous computer experience).

You start to program your VZ300 by writing and playing games, so you can have fun while you're learning!

In Chapter One, you'll take your first steps in programming, and discover that you already know a number of words in the BASIC programming language.

By the end of Chapter Three, you'll have run several complete programs, and be in full control of the material your VZ300 is printing on the screen.

You go on to roll dice with the VZ's random number generator; loop the loop with FOR/NEXT loops; read, write and compare data; and use sound and graphics to enhance your programs. After this, you'll take real control of your VZ300 with programming commands such as PEEK and POKE.

Finally, you'll discover how to create your own moving graphics and write animated arcade games!

This book includes a generous collection of VZ300 game and demonstration programs, including MASTERMIND, STAR COLONY, REACTION TEST, GOLF, THE PHANTOM COMPOSER, SPIROGRAPH, LISSAJOUS CURVES, TRAPPER, MESOZOIC ATTACK and V-WING SPACE BATTLE.
Come on in, the programming's fine!

X-7326